

Event Driven Programming

DCAP211

Edited by:
Kumar Vishal



L OVELY
P ROFESSIONAL
U NIVERSITY



Event Driven Programming

Edited By
Kumar Vishal

Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

SYLLABUS

Event Driven Programming

This course introduce computer programming using VISUAL BASICS. Emphasis is placed on Event Driven Programming methods so that he may be able to construct attractive and user friendly interfaces.

Sr. No.	Topics
1.	Integrated Development Environment: Features and applications of VB – concept of integrated development environment (IDE) – project application like standard Exe, Active X EXE, Data reports, methods and events, event procedures
2.	Understanding Controls and Control Events: Design aspects of VB forms, Elements of user Interface, properties of controls–textbox, label, command button, check box and list box. Combo box, frames, option button, picture box. Forms properties and Form load event
3.	VB Programming Fundamentals: Datatypes, Variables, variable declaration, fixed length vs variable length string, implicit and explicit variable declaration, variable scope and lifetime. Operators-math operator, concatenation operator, logical operator
4.	Control structures and Arrays: If-else ,do-while ,while-went, for-next, select-case. Defining Array, using redim and preserve function in array and control arrays
5.	Understanding Functions and Procedures: Procedures and functions. Modules
6.	Menus and Dialog Boxes : Designing menus using menu editor, assigning access keys and shortcut keys, separating menu items. creating pop up menus, controlling menus at runtime-enabling, disabling, invisible menu commands. Modal and Modless dialog boxes and predefined dialog boxes
7.	Data Base Fundamentals and Connectivity Options: Data Control and Data bound control, creating database using visual database manager, connectivity of vb with ms-access, sql server
8.	Data Base Access using ADO Data Control : Using Ado control –Adodc and Adodb-recordset properties, connection string properties, displaying data from database in grids and data bound controls
9.	Working with Reports: Data environment, creating various types data report using grid and data bound controls
10.	Buliding Small Application: Using Splash Screens, Timers, Images

CONTENT

Unit 1:	The Integrated Development Environment <i>Kumar Vishal, Lovely Professional University</i>	1
Unit 2:	Working of VB Applications <i>Kumar Vishal, Lovely Professional University</i>	12
Unit 3:	Understanding Controls and Control Events <i>Kumar Vishal, Lovely Professional University</i>	32
Unit 4:	VB Programming Fundamentals <i>Kumar Vishal, Lovely Professional University</i>	77
Unit 5:	VB String and Operators <i>Kumar Vishal, Lovely Professional University</i>	92
Unit 6:	VB Control Structure <i>Kumar Vishal, Lovely Professional University</i>	121
Unit 7:	Arrays in Visual Basic <i>Kumar Vishal, Lovely Professional University</i>	165
Unit 8:	Understanding Function and Procedure <i>Kumar Vishal, Lovely Professional University</i>	182
Unit 9:	Menus and Dialog Boxes <i>Anil Sharma, Lovely Professional University</i>	201
Unit 10:	Database Fundamentals and Connectivity Option <i>Anil Sharma, Lovely Professional University</i>	220
Unit 11:	Data Base Access <i>Anil Sharma, Lovely Professional University</i>	244
Unit 12:	ADO Data Control <i>Anil Sharma, Lovely Professional University</i>	258
Unit 13:	Working With Reports <i>Anil Sharma, Lovely Professional University</i>	278
Unit 14:	Building Small Application <i>Anil Sharma, Lovely Professional University</i>	299

Unit 1: The Integrated Development Environment

CONTENTS

Objectives
Introduction
1.1 Visual Basic
1.2 Features of Visual Basic
1.2.1 Structure of a Visual Basic Application
1.3 Application of VB
1.4 The Integrated Development Environment
1.4.1 The Menu Bar
1.4.2 The Toolbar
1.5 Summary
1.6 Keywords
1.7 Self Assessment Questions
1.8 Review Questions
1.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand visual basic
- Explain feature of visual basic
- Discuss application of visual basic
- Understand integrated development environment.

Introduction

An integrated development environment (IDE) (also known as **integrated design environment** or **integrated debugging environment**) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of:

- a source code editor
- a compiler and/or an interpreter
- build automation tools
- a debugger

Sometimes a version control system and various tools are integrated to simplify the construction of a GUI. Many modern IDEs also have a class browser, an object inspector, and a class hierarchy diagram, for use with object-oriented software development.

1.1 Visual Basic

Visual Basic is a tool that allows you to develop Windows (Graphic User Interface - **GUI**) applications. The applications have a familiar appearance to the user. Visual Basic is **event-driven**, meaning code remains idle until called upon to respond to some event (button pressing, menu selection,). Visual Basic is governed by an event processor. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.

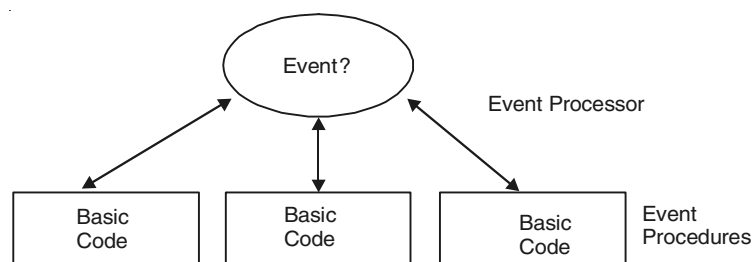


Figure 1.1: Event-driven.

1.2 Features of Visual Basic

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

Visual Basic 6.0 Versus Other Versions of Visual Basic

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

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

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

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

Visual Basic 6.0 - some identified new features of Visual Basic 6.0

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

16 Bits Versus 32 Bits

Applications built using the Visual Basic 3.0 and the 16 bit version of Visual Basic 4.0 will run under Windows 3.1, Windows for Workgroups, Windows NT, or Windows 95 Applications built using the 32 bit version of Visual Basic 4.0, Visual Basic 5.0 and Visual Basic 6.0 will only run with Windows 95 or Windows NT (Version 3.5.1 or higher). In this class, we will use

Visual Basic 6.0 under Windows 95, recognizing such applications will not operate in 16 bit environments.

1.2.1 Structure of a Visual Basic Application

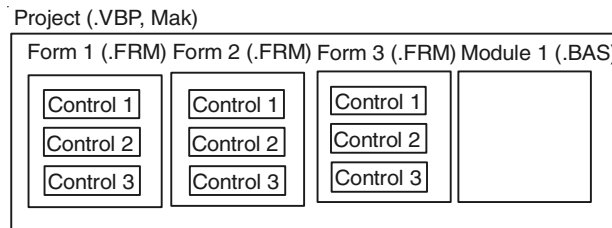


Figure 1.2: Structure of a Visual Basic Application.

1.3 Application OF VB

Visual Basic Applications (VBA): This is an implementation of Microsoft's event-driven programming language, Visual Basic 6 and its associated integrated development environment (IDE), which are built into most Microsoft Office applications. VBA enables developers to build user defined functions, automate processes and access Win 32 and other low-level functionality through DLLs. It was also built into office applications apart from version 2008 for Apple's Mac OS X, other Microsoft applications such as Microsoft MapPoint and Microsoft Visio as well as being at least partially implemented in some other applications such as AutoCAD, Word Perfect and ArcGIS. It supersedes and expands on the capabilities of earlier application-specific macro programming languages such as Word's Word Basic. It can be used to control many aspects of the host application, including manipulating user interface features, such as menus and toolbars, and working with custom user forms or dialog boxes. VBA can also be used to create import and export filters for various file formats, such as ODF.

As its name suggests, VBA is closely related to Visual Basic and uses the Visual Basic Runtime, but can normally only run code within a host application rather than as a standalone application. It can, however, be used to control one application from another using OLE Automation. For example, it is used automatically to create a Word report from Excel data, in turn automatically collected by Excel from Polled observation sensors. The VBA IDE is reached from within an office document by pressing the key sequence Alt+F11.

VBA is functionally rich and flexible but it does have some important limitations, such as restricted support for function pointers which are used as callback functions in the Windows API. It has the ability to use (but not create) (Active X/COM) DLLs, and later versions add support for class modules.

1.4 The Integrated Development Environment

The Visual Basic IDE appears in Figure, and as a Visual Basic programmer, this is where you'll spend most of your programming time. If you are not already familiar with the parts of the IDE, you will be in time.

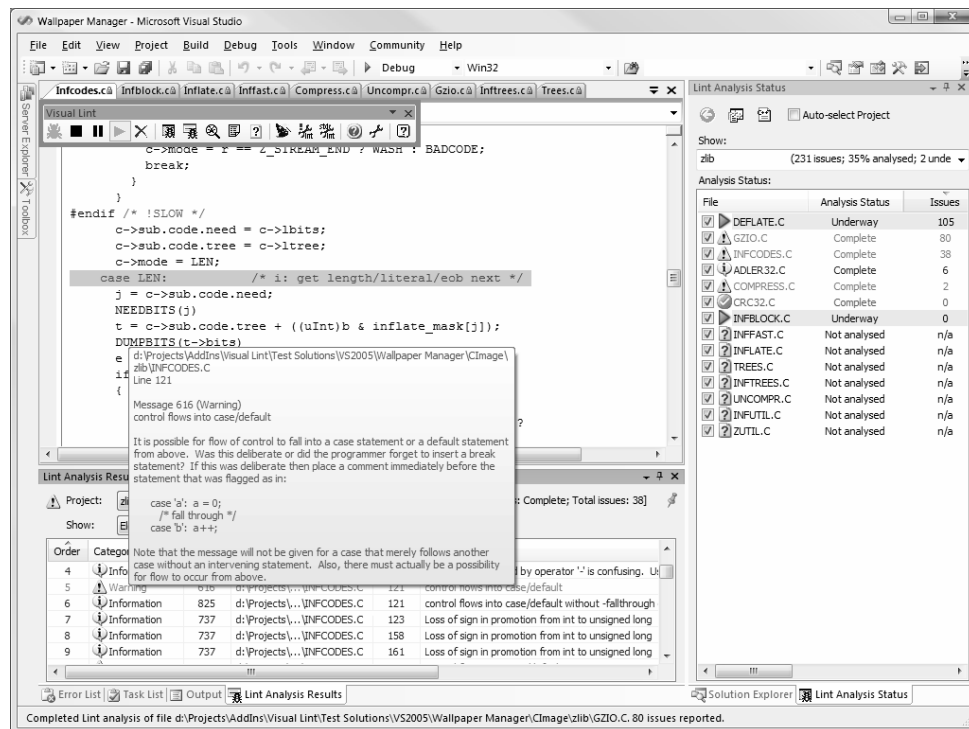


Figure 1.3: The Visual Basic Integrated Development Environment.

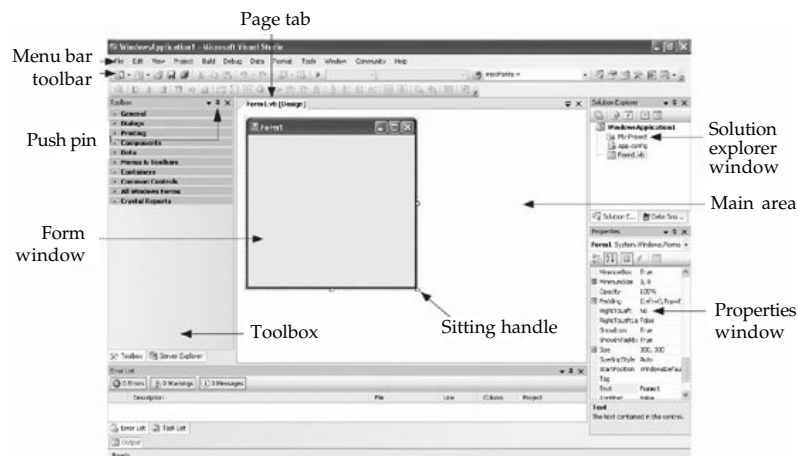


Figure 1.4: The Visual Basic Programming Environment.

The Visual Basic IDE has three distinct states: Design, Run, and Debug. The current state appears in Visual Basic's title bar. This chapter concentrates on the Design state. We'll cover the Debug state later in the book. (In the Run state, Visual Basic is in the background while your program runs.) It's the Design state that's become complex over the years, and we'll lay it bare in this unit.

The IDE is composed of these parts:

- " The menu bar
- " The toolbar
- " The Project explorer
- " The Properties window
- " The Form Layout window
- " The toolbox
- " Form designers
- " Code windows

We'll take a look at all of these parts in this overview.

1.4.1 The Menu Bar

The menu bar presents the Visual Basic menus. Here's a list of those menus and what they do:

- "*File* File handling and printing; also used to make EXE files
- "*Edit* Standard editing functions, undo, searches
- "*View* Displays or hides windows and toolbars
- "*Project* Sets project properties, adds/removes forms and modules, and Adds/removes references and components
- "*Format* Aligns or sizes controls
- "*Debug* Starts/stops debugging and stepping through programs
- "*Run* Starts a program, or compiles and starts it
- "*Tools* Adds procedures, starts the Menu Editor, sets IDE options
- "*Add-Ins* Add-in manager, lists add-ins like Application Wizard and API Viewer
- "*Window* Arranges or selects open windows
- "*Help* Handles Help and the About box



Notes

One important job of the File menu is to create EXE files for your program. When you run a program from the Run menu, no EXE file is created; if you want to run the program outside of Visual Basic, you must create that EXE file, and you do that with the File menus Make ProjectName.exe item (where Project Name is the name you've set for the project).

We'll see a great deal more about these menus and the items they contain in the Immediate Solutions section of this chapter.

1.4.2 The Toolbar

The main Visual Basic toolbar appears in Figure 1.5. This toolbar contains buttons matching popular menu items, as you can see in Figure 1.5; clicking the button is the same as selecting a menu item and can save you some time.

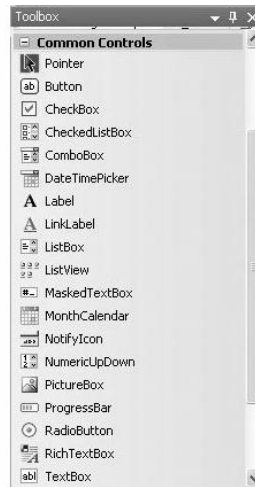


Figure 1.5: The Main Visual Basic Toolbar.

Besides the main toolbar, you can also display other dockable toolbars in Visual Basic: the Debug, Edit, and Form Editor toolbars. To display one of these toolbars, just select it using the Toolbars item in the View menu; the toolbar appears free-floating at first, but you can dock it as you like in the IDE.



Notes

If you are unsure what a particular tool in the toolbar does, just rest the mouse over it. A tool tip (a small yellow window displaying text) will display the tools purpose.

The **Properties Window** is used to establish initial property values for objects. The drop-down box at the top of the window lists all objects in the current form. Two views are available: Alphabetic and Categorized. Under this box are the available properties for the currently selected object.

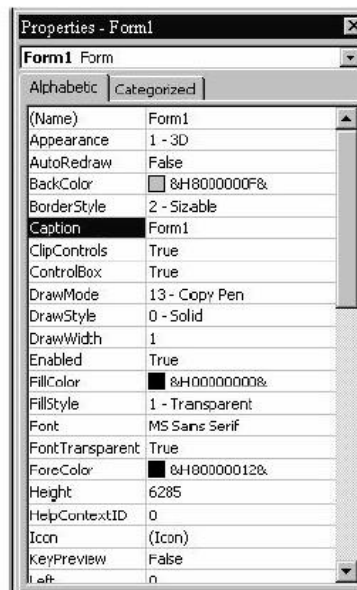


Figure 1.6: Property Windows.

The **Form Layout Window** shows where (upon program execution) your form will be displayed relative to your monitor's screen:

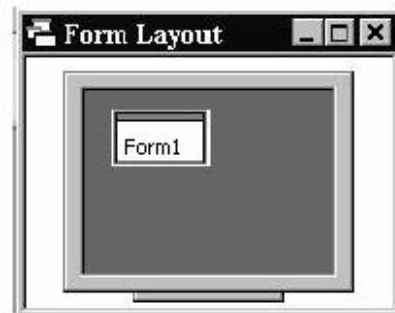


Figure 1.7: Form Layout.

The **Project Window** displays a list of all forms and modules making up your application. You can also obtain a view of the **Form** or **Code** windows (window containing the actual Basic coding) from the Project window.

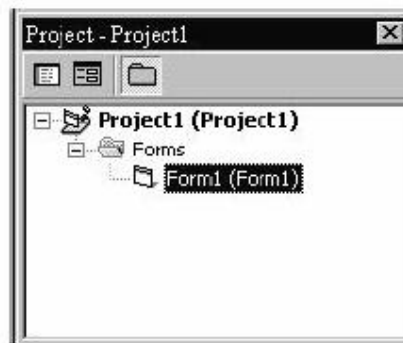


Figure 1.8: Project Explore.

As mentioned, the user interface is 'drawn' in the form window. There are two ways to place controls on a form:

1. Double-click the tool in the toolbox and it is created with a default size on the form. You can then move it or resize it.
2. Click the tool in the toolbox, then move the mouse pointer to the form window. The cursor changes to a crosshair. Place the crosshair at the upper left corner of where you want the control to be, press the left mouse button and hold it down while dragging the cursor toward the lower right corner. When you release the mouse button, the control is drawn.

To **move** a control you have drawn, click the object in the form window and drag it to the new location. Release the mouse button.

To **resize** a control, click the object so that it is select and sizing handles appear. Use these handles to resize the object.

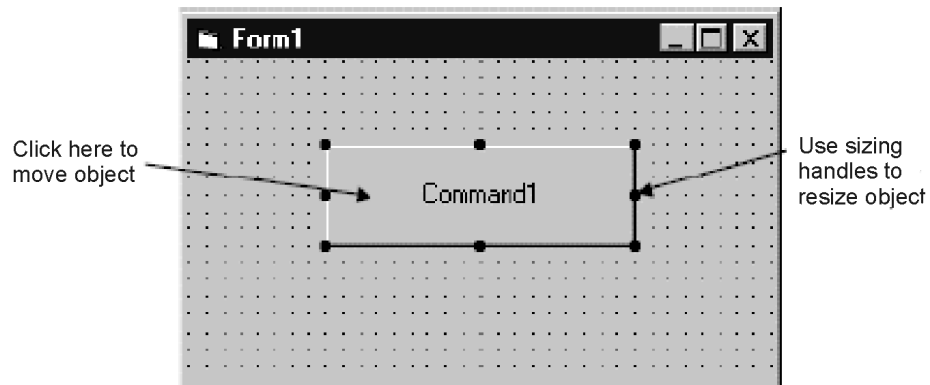


Figure 1.9: Command Button.

Form Designers and Code Windows

The last parts of the IDE that we will take a look at in our overview are form designers and code windows, which appear in the center of Figure 1.9. (The form designer displays the current form under design, complete with command button, and the code window displays the code for the **Command1_Click()** procedure.)



Case Study

Success Story of Visual Basic

The History of Visual Basic dates back to 1991 when VB 1.0 was introduced. The core of Visual Basic was built on the older BASIC language, which was a popular programming language throughout the 1980s.

Alan Cooper had developed a drag-and-drop interface in the late 1980s, Microsoft approached him and asked his company, Tripod, to develop the concept into a form building application. Tripod developed the project for Microsoft. It was called Ruby and it did not include a programming language at all. Microsoft decided to bundle it with the BASIC programming language, creating Visual Basic. Ruby also provided the ability to load dynamic link libraries containing additional controls (then called 'gizmos'), which later became the VBX interface.

Visual Basic 1.0 for Windows was released in May 1991 at a trade show in Atlanta and Georgia. Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved. Notably, forms became core objects, thus laying the foundational concepts of class modules as were later offered in VB4.

Visual Basic 3.0 was released in 1993 and came in Standard and Professional versions. VB3 included version 1.1 of the Microsoft Jet Database Engine that could read and write Jet (or Access) 1.x databases. Visual Basic 4.0 was released in August 1995. It was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to

write non-GUI classes in Visual Basic. While previous Versions of Visual Basic had used VBX controls, Visual Basic now used OLE controls (with files names ending in .ocx) instead. These were later to be named Active X controls.

With version 5.0 release in February 1997, Microsoft released Visual Basic exclusively for 32-bit versions of Windows. Programmers who preferred to write 16-bit programs were able to import programs written in Visual Basic 4.0 to Visual Basic 5.0 and Visual Basic 5.0 programs could easily be converted with Visual Basic 4.0. Visual Basic 5.0 also introduced the ability to create custom user controls, as well as the ability to compile to native Windows executable code, speeding up calculation-intensive code execution. A free, downloadable Control Creation Edition was also released for creation of Active X controls.

Visual Basic 6.0 released in mid 1998 improved in a number of areas including the ability to create web-based applications. VB6 has been the most successful version in the history of Visual of Basic, it has entered Microsoft's 'non-supported phase' as of March 2008. Although the development environment is no longer supported, the runtime is supported on Windows Vista, Windows Server 2008 and Windows 7.

Mainstream Support for Microsoft VB 6.0 ended on March 31,2005. Extended support ended in March 2008. In response, the Visual Basic user community expressed its grave concern and lobbied users to sign a petition to keep the product alive, Microsoft has so far refused to change their position on the matter.

Questions

1. Differentiate between first and last version of Visual Basic.
2. Explain the concept of Active control.

1.5 Summary

- Visual basic to is a tool that allows you to develop window (Graphic user interface application.
- Visual basic is an implement of microsoft event-driven programming language.
- The visual basic IDE has three distinct states: Design, Run and debug.

1.6 Keywords

Event-Driven: VB is event driven, means code remains idle until called upon to respond to some event. VB is governed by an event.

Graphical user interface (GUI): In computing a **graphical user interface (GUI**, sometimes pronounced gooey,) is a type of user interface that allows users to interact with electronic devices with images rather than text commands.

Integrated Development Environment: An **integrated development environment (IDE)** (also known as **integrated design environment** or **integrated debugging environment**) is a software application that provides comprehensive facilities to computer programmers for software development.

The form layout window: It displays the initial position and relative size of the current form shown in the form window.

The properties window: It displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.

1.7 Self Assessment Questions

1. An integrated development environment (IDE) is also known as:
 - (a) Integrated Design Environment
 - (b) Integrated Debugging Environment
 - (c) Both (a) and (b)
 - (d) None of these
2. Visual Basic is a tool that allows you to develop.
 - (a) Windows Applications
 - (b) Visual Applications
 - (c) General Applications
 - (d) All of these
3. Visual Basic 3.0 was released in
 - (a) 1990
 - (b) 1991
 - (c) 1995
 - (d) 1993
4. Applications built using the Visual Basic 3.0 and the 16-bit version of Visual Basic 4.0 will run under:
 - (a) Windows 3.0
 - (b) Windows 3.1
 - (c) Windows 3.2
 - (d) Windows 3.5
5. The _____ is used to establish initial property values for objects.
 - (a) Project window
 - (b) Properties window
 - (c) Form Layout window
 - (d) All of these

1.8 Review Questions

1. Create a text box containing the words "PLAY IT, SAM" in blue letters.
2. What is the difference between ADODC and ADODB?
3. How many events in VB6 form?
4. Does VB support OOPS Concepts? Explain.
5. What is Marshalling?
6. What is the difference between form file and binary file (difference between frm and frx)?

Answers for Self Assessment Questions

1. (c)
2. (a)
3. (d)
4. (b)
5. (b)

1.9 Further Readings



Books

G. Cornell, "Visual Basic 6" Tata McGraw-Hill, 1998.

Deitel & Deitel, T.R. Nieto, "Visual Basic 6, How to program"
Prentice Hall of India, 1999.



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

Unit 2: Working of VB Applications

CONTENTS

Objectives

Introduction

2.1 Working of VB Applications

2.1.1 Active Exe

2.1.2 Standard EXE Application

2.1.3 Creating a Data Report

2.1.4 Accessing the Data Report

2.2 Method and Events

2.3 Event Procedures

2.4 Summary

2.5 Keywords

2.6 Self Assessment Questions

2.7 Review Questions

2.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand working of VB applications
- Explain Method and event in visual basic
- Discuss event procedures.

Introduction

Visual Basic is a high level programming language which evolved from the earlier DOS version called BASIC. BASIC means, Beginners' All-Purpose Symbolic Instruction Code. It is a very easy programming language to learn. The code look a lot like English Language. Different software companies produced different versions of Basic, such as Microsoft QBASIC, QUICK BASIC, GW BASIC, IBM BASICA and so on. However, people prefer to use Microsoft Visual Basic today, as it is a well developed programming language and supporting resources are available everywhere. Now, there are many versions of VB exist in the market, the most popular one and still widely

used by many VB programmers is none other than Visual Basic 6. We also have VB. Net, VB 2005, VB 2008 and the latest VB 2010. Both VB 2008 and VB 2010 are fully object oriented programming (OOP) language.

VISUAL BASIC is a VISUAL and events driven Programming Language. These are the main divergence from the old BASIC. In BASIC, programming is done in a text-only environment and the program is executed sequentially. In VB, programming is done in a graphical environment. In the old BASIC, you have to write program code for each graphical object you wish to display it on screen, including its position and its color. However, in VB, you just need to drag and drop any graphical object anywhere on the form, and you can change its color any time using the properties windows.

2.1 Working of VB Applications

The fundamental aspects of developing a standard exe file, as well as an ActiveX EXE file. We also demonstrate the techniques for interacting between Standard EXE and ActiveX EXE programs by passing parameter values.

2.1.1 Active Exe

We develop a simple ActiveX EXE with just one form. The name of the ActiveX component is clsActivexEx. The example is chosen to demonstrate the use of ActiveX EXE in conjunction with Standard EXE.



Figure 2.1: Activex-1, Dialog Box Showing the Type of Project to Open.

The selection (number/alphabet but not both) is passed on to the ActiveX EXE. Now, in the ActiveX EXE, you select specific number (say "2") or alphabet (say "A"). That specific number or alphabet is again passed back to the Standard EXE. It is similar to, say, the Standard EXE selects either birds or animals. If the Standard EXE chooses birds, then the ActiveX EXE can choose any one of a given set of birds. It doesn't have the option to choose any animal (because the Standard

EXE already chosen birds). The selected bird is passed back to the Standard EXE. The example makes it clear as you read on.

1. Open Visual Basic 6.0—it should be on your start menu if you have just installed it
Programs -> Microsoft Visual Basic 6.0 -> Microsoft Visual Basic 6.0 Open an instance of Visual Basic Choose ActiveX EXE and select Open.

Go to the properties of class and change its name property to clsActivexEx. Add a new Form by using Add Form menu item in the project menu. Change the name property of form to frmActivexEx and the caption to "Form From ActiveX Exe".

Include the following controls on the form:

Object	Property	Setting
Command button	Name	cmd SelAnd Close
	Caption	Select and close
Combo Box	Name	cboNumbers
Combo Box	Name	cboAlphabets
Label	Caption	Numbers:
Label	Caption	Alphabets:

Open the project properties dialog box by going to Project -> ActivexEx Properties.

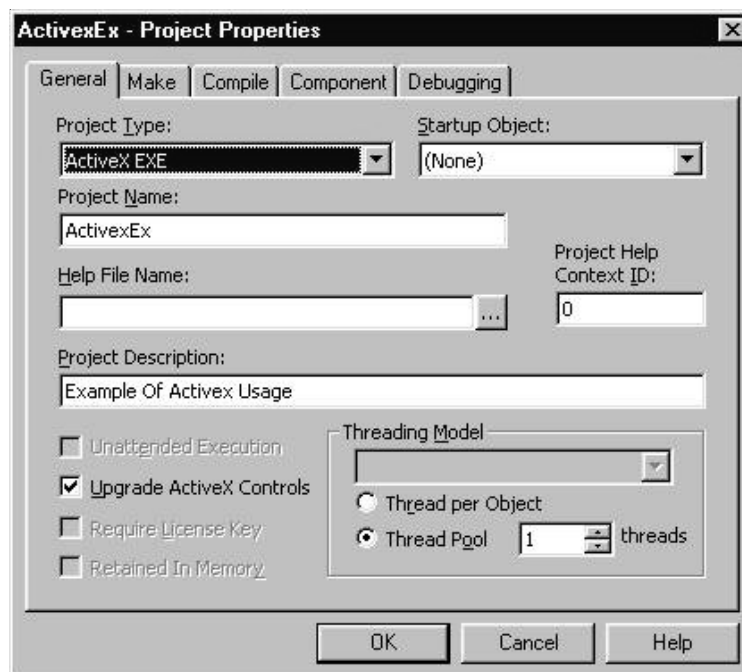


Figure 2.2: ActiveX-2 Project Properties Dialog Box.

In the above figure Project type specifies the type of project that is being developed, in this case the project is of type ActiveX Exe, other project types can be seen by clicking on the combo box.

“Project Name” specifies the name of the component type library.

You include the ActiveX reference at a later stage in to the Standard EXE. Note that the “Project Description” is the one that appears in “Add Reference” dialog box when its corresponding component is added.

In this case, startup object is set to “None”, as there is no initialization required for this project.

Go to the component tab and check for the following:

“Version Compatibility” by default is of type “Project Compatibility” for all ActiveX Project types.

“Start Mode” is of type “ActiveX Component” by default. Select the “Standalone” option button, only if you intend to start the application as a standalone.

Now, you write the required code. We have added a new Module for placing our code.

Go to Project -> Add Module.

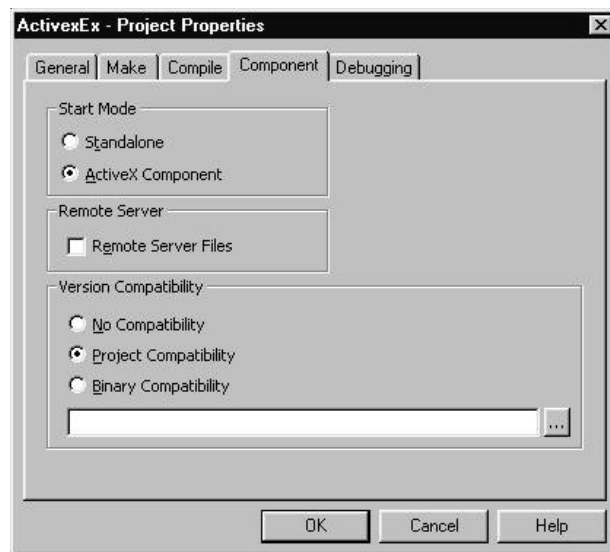


Figure 2.3: ActiveX-3, Project Properties Dialog Box.



Did u know?

Don't Use the Initialize Event to Set an ActiveX Control's Default Property Values. If you put code to initialize properties to their default values in your custom ActiveX control's Initialize event instead of in the InitProperties event, then you will have some very frustrated developers on your hands. Your default values will override the values the developer has assigned at design time every time the developer runs an application using your control.

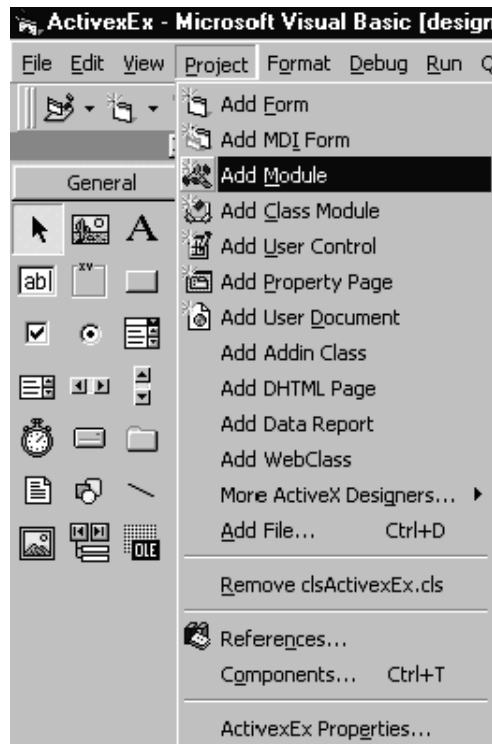


Figure 2.4: ActiveX-4; Project Menu that Used for Adding a New Module.

Refer to appendix-A for code to be included in Class Module.

Refer to appendix-B for code to be included in Form.

Refer to appendix-C for code to be included in Standard Module.

Use ctrl-F5 to compile and make sure that there are no compilation errors.

From the file menu select Make Activex_ex.exe to make an exe file of this project (which is called reference executable) as shown below:



Did u know?

Single-Threaded ActiveX Controls Cause Problems in Multithreaded Clients
If a client application is multithreaded; a single-threaded ActiveX control can cause problems for the multithreaded client. In fact, if a multithreaded client application is being written in VB, VB will not even allow programmers to use a single-threaded ActiveX control in the project.

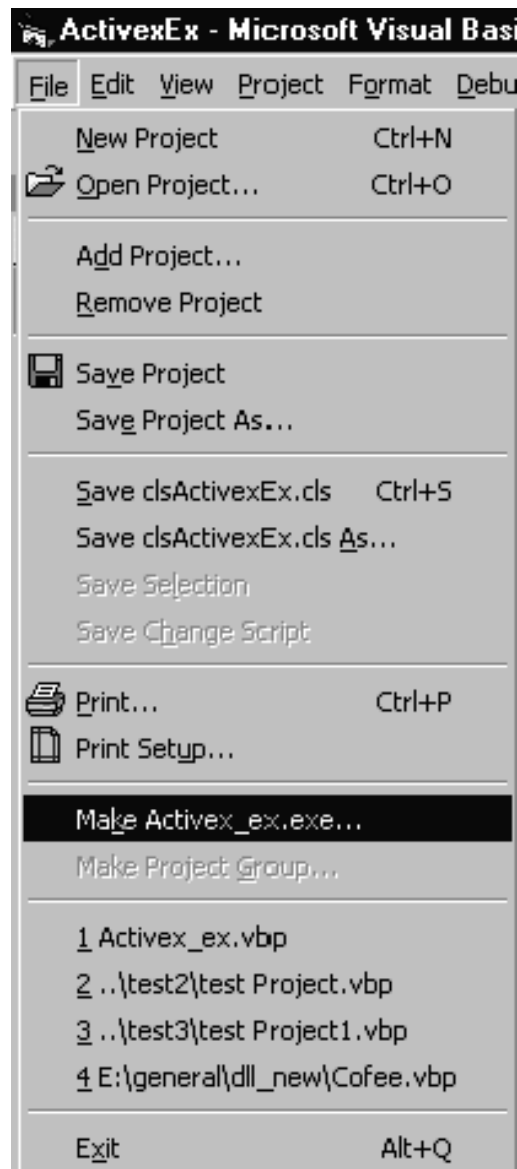


Figure 2.5: ActiveX-5; Menu Item Showing Make .exe file.

2.1.2 Standard EXE Application

1. Open Visual Basic 6.0—it should be on your start menu if you have just installed it (Programs -> Microsoft Visual Basic 6.0 -> Microsoft Visual Basic 6.0) Open another instance of Visual Basic, select the "Standard EXE" and click "Open".

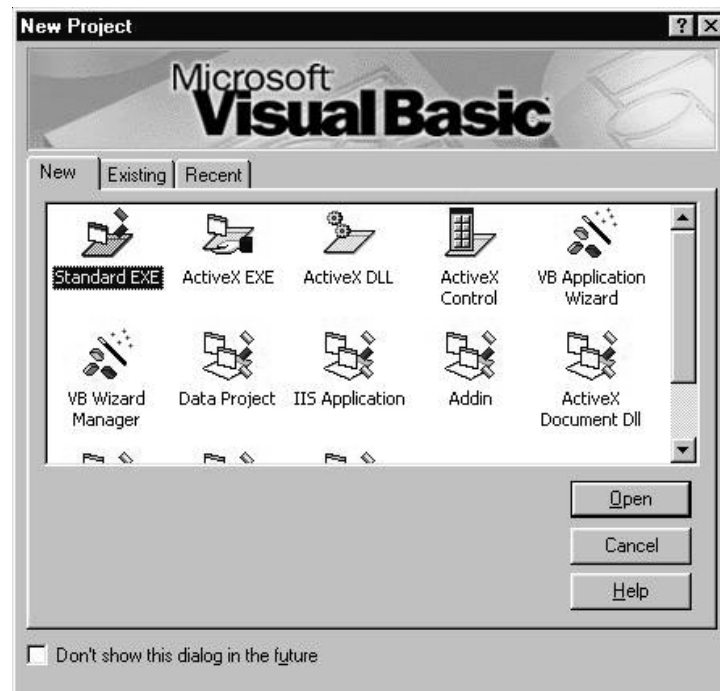


Figure 2.6: ActiveX-6;Dialog Box Showing the Type of Project to Open.

Change the name property of form to frmTestActivex and caption to “Test form”
Include the following controls on the form:

Object	Property	Setting
Command button	Name Caption	CmdShowSelVal Show Selected Value in ActiveX Exe
Command button	Name Caption	cmdCallActivex Show Form From ActiveX Exe
Option button	Name Caption	opt Numbers Choose Numbers in ActiveX Exe
Option button	Name Caption	OptAlphabets Choose Alphabets in ActiveX Exe

Select References from Project menu by going to “Project -> References” as shown in the figure below:

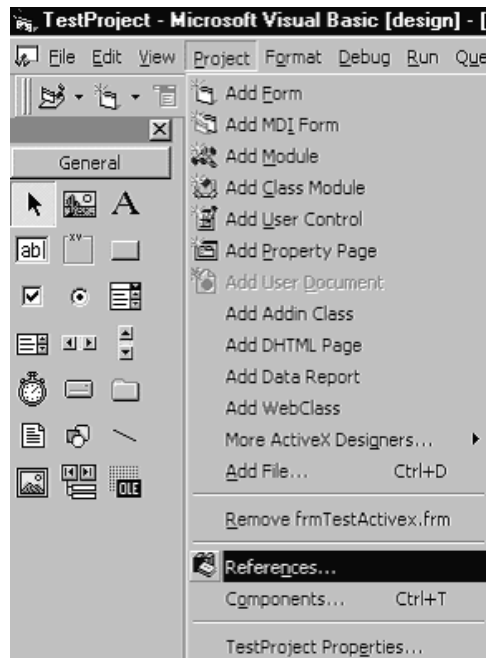


Figure 2.7: ActiveX-7; References Dialog Box.

On selecting References Menu item a dialog box as shown below will be opened.

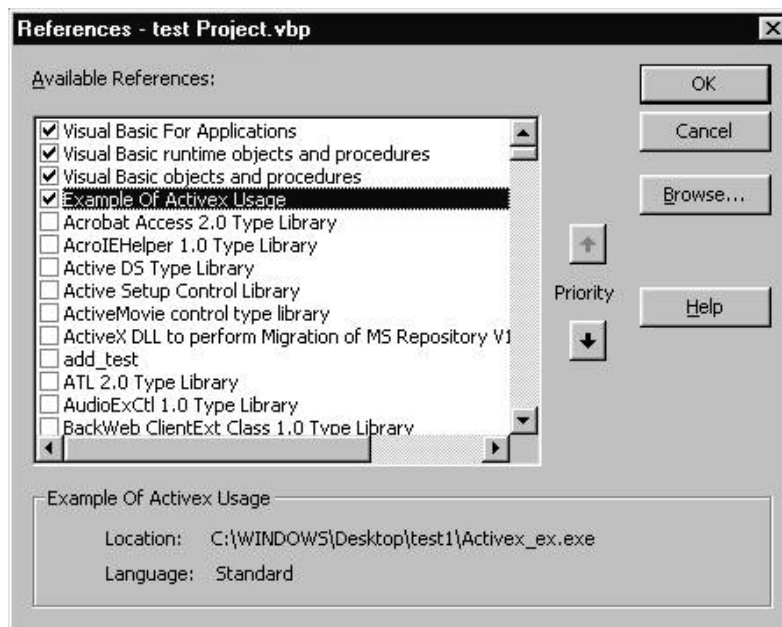


Figure 2.8: ActiveX-8; References Dialog Box.

Select the ActiveX Exe made in the above project and click on OK. In case you don't see the ActiveX component that you had just made, use "Browse" button locate and select the component.

Refer to appendix-D for code to be included in Form.

5.0 Interacting with ActiveX exe from Standard exe and vice versa.

To demonstrate how the Standard EXE interacts with the ActiveX EXE, run the test project. Assuming that the code is entered properly, a form as shown below is displayed:

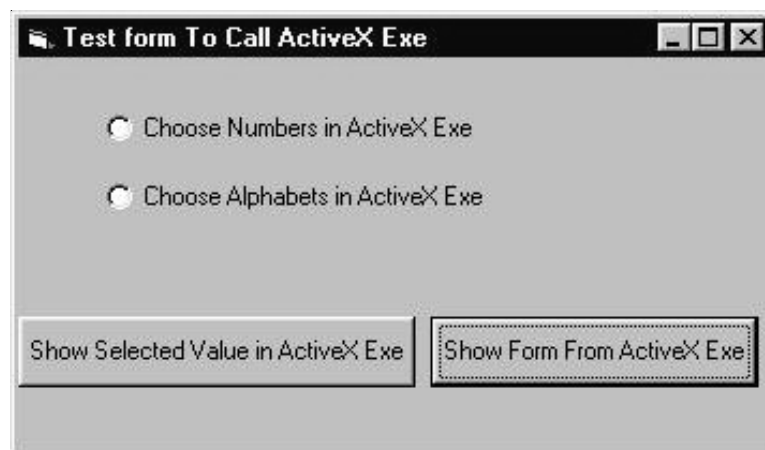


Figure 2.9: ActiveX-9; Test Form.

Select either of the options on the form and click on "show form from ActiveX Exe" Button.

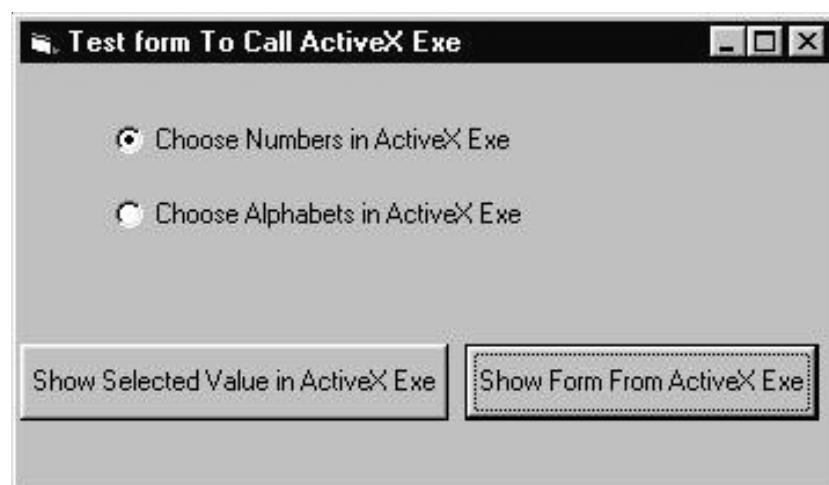


Figure 2.10: ActiveX-10; Test Form Selection.

On selecting and clicking the button opens a form in AcctivexEx Component as shown in Figure 2.10.

In this case, Alphabets Drop-Down box is disabled as "Choose Numbers in ActiveX Exe" has been selected.

Here a value is passed from Standard Exe to ActiveX Exe based on the option button we selected and its corresponding drop-down box is enabled in ActiveX.

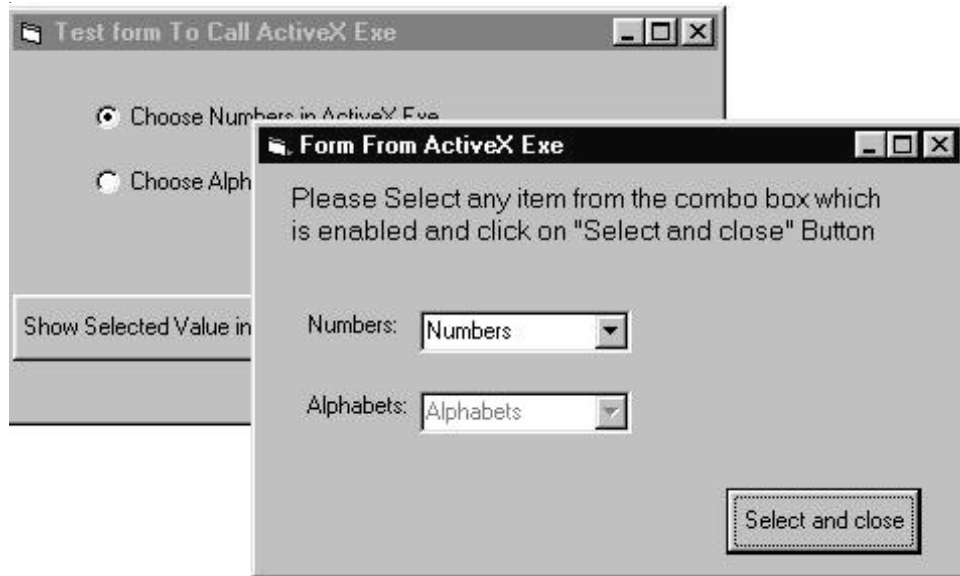


Figure 2.11: ActiveX-11; ActiveX EXE Call.

Select any number from the drop-down box (Numbers Drop-down box in this case) and click on "Select and close" Button

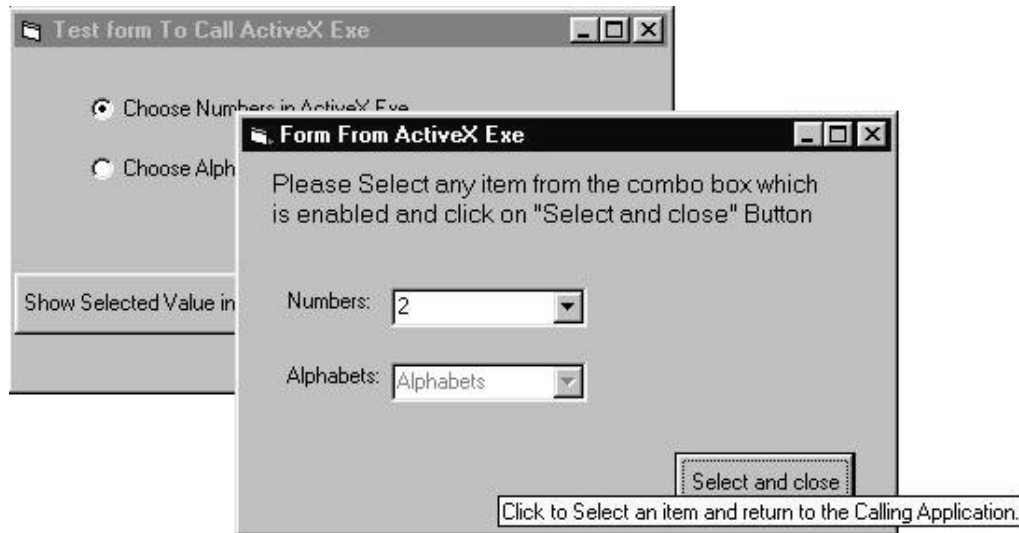


Figure 2.12: ActiveX-12; Here the Control is passed to the ActiveX EXE Application.

On clicking the "Select and close" button the form in ActiveX Exe is closed, and the control is passed back to the Main (Standard EXE file). However, note that the ActiveX EXE is still running in the background. It is possible to close ActiveX EXE, if so desired. In our example, it would be closed only when the Main application is closed.

Now click on “Show Selected Value in ActiveX Exe” button to see the item that is selected in ActiveX Exe. Here the number/alphabet selected in ActiveX passed to Standard Exe.

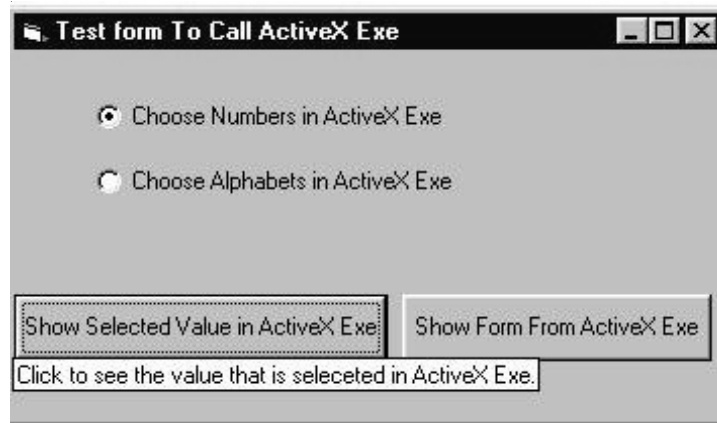


Figure 2.13: ActiveX-13; Control is passed back to the Main (Standard EXE). Below shows the item that was selected in ActiveX Exe.

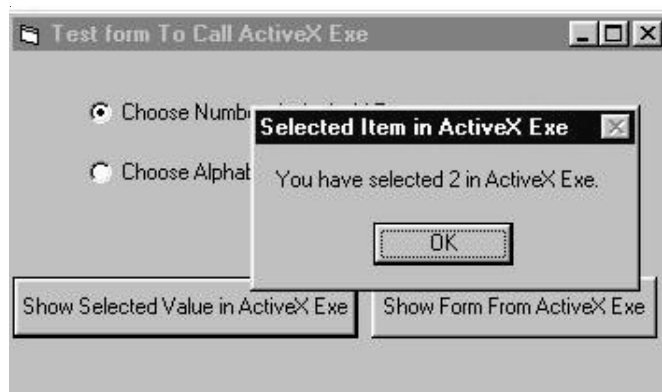


Figure 2.14: The selected number is displayed in the Main (Standard EXE).

As you can see, in the above example, we have selected “numbers” in the Standard EXE, and later selected “2” in the ActiveX EXE. The number “2” is passed on to the Standard EXE upon request.



Notes

This is only a brief example provided to demonstrate the concept. Practically, the ActiveX EXE/DLL could be quite large, such as a calculator. The calling program may be a Text Editor.

2.1.3 Creating a Data Report

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

1. On the Project menu, click Add Data Report and one will be added to your project. If this item is not on the menu, click Components. Click the Designers tab, and choose Data Report and click OK to add the designer to your menu.
2. Set the following properties for the report:

Name - rptPhone

Caption - Phone Directory

DataSource - denPhone (your phone data environment - choose, don't type)

DataMember - PhoneList (the table name - choose don't type)

3. Right-click the **Data Report** and click **Retrieve Structure**. This establishes a report format based on the **Data Environment**.
4. There are five sections to the data report: a Report Header, a Page Header, a Detail section, a Page Footer, and a Report Footer. The headers and footers contain information you want printed in the report and on each page. To place information in one of these regions, right-click the selected region, click Add Control, and then choose the control you wish to place. These controls are called data report controls and properties are established just like you do for usual controls. Try adding some headers.
5. The Detail section is used to layout the information you want printed for each record in your database. We will place two field listings (Name, Phone) there. Click on the Name tab in the Data Environment window and drag it to the Detail section of the Data Report. Two items should appear: a text box Name and a text box Name (PhoneList). The first text box is heading information. Move this text box into the Page Header section. The second text box is the actual value for Name from the PhoneList table. Line this text box up under the Name header. Now, drag the Phone tab from the Data Environment to the Data Report. Adjust the text boxes in the same manner. Our data report will have page headers Name and Phone. Under these headers, these fields for each record in our database will be displayed. When done, the form should look something like this:



Figure 2.15: Data Report Creating.

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

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

2.1.4 Accessing the Data Report

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

3. Use this code in cmdReport_Click:

```
Private Sub cmdReport_Click()  
    rptPhone.Show  
End Sub
```

4. This uses the Show method to display the data report.
5. Save the application and run it. Click the Show Report button and this should appear:



Figure 2.16: Data Report Accessing.

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



Who Is the “User” of an ActiveX Control? Much of the documentation on ActiveX custom controls employs the word “user” loosely, sometimes meaning the end user of the application where your control is sited and, at other times, meaning the developer who is programming with your control.

2.2 Method and Events

A method is an action that can be performed on objects. For example, a cat is an object. Its properties might include long white hair, blue eyes, 3 pounds weight etc. A complete definition of cat must only encompass on its looks, but should also include a complete itemization of its activities. Therefore, a cat’s methods might be move, jump, play, breath etc. Similarly, in object oriented programming, a method is a connected or built-in procedure, a block of code that can be invoked to impart some action on a particular object. A method requires an object to provide them with a context. For example, the word Move has no meaning in Visual Basic, but the statement, Text1.Move 700, 400 performs a very precise action. The TextBox control has other associated methods such as Refresh, SetFocus, etc.

- The Refresh method enforces a complete repaint of the control or a Form. For example, Text1.Refresh refreshes the TextBox.
- The Setfocus method moves the focus on the control. For Example Text1.SetFocus sets the focus to TextBox control Text1.

2.3 Event Procedures

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

```
Private Sub Form_Load()
```

```
....statement block..
```

```
End Sub
```

Event Procedures acquire the declarations as Private by default.



Case Study

How to create the Crystal Report in Visual Basic?

To access the Crystal Report Designer, a Start menu group and shortcut will have been automatically created.

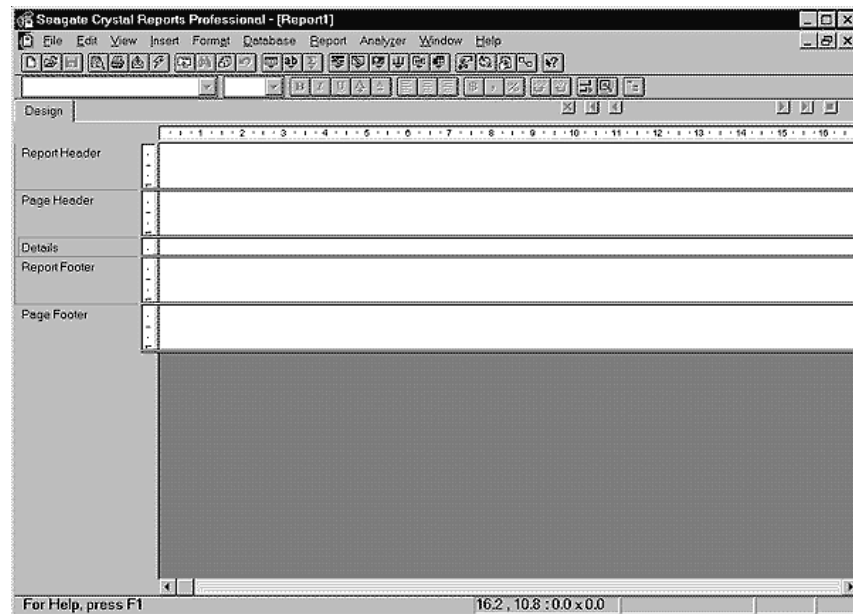
Lets have a quick look at the Designer

Initially there is one tab on the report - Design - when you select the print preview to see how the report looks with data in, another tab named Preview will appear.

While you are in design mode you can draw and arrange the data fields on the report. This is done in a similar way to creating controls on a form in VB.

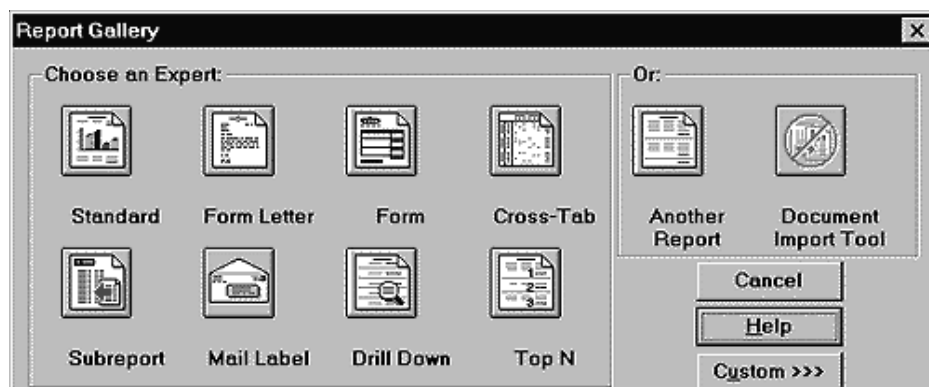
For the purposes of this tutorial we will be creating a simple report from the sample database BIBLIO.

Start the Crystal Reports Designer

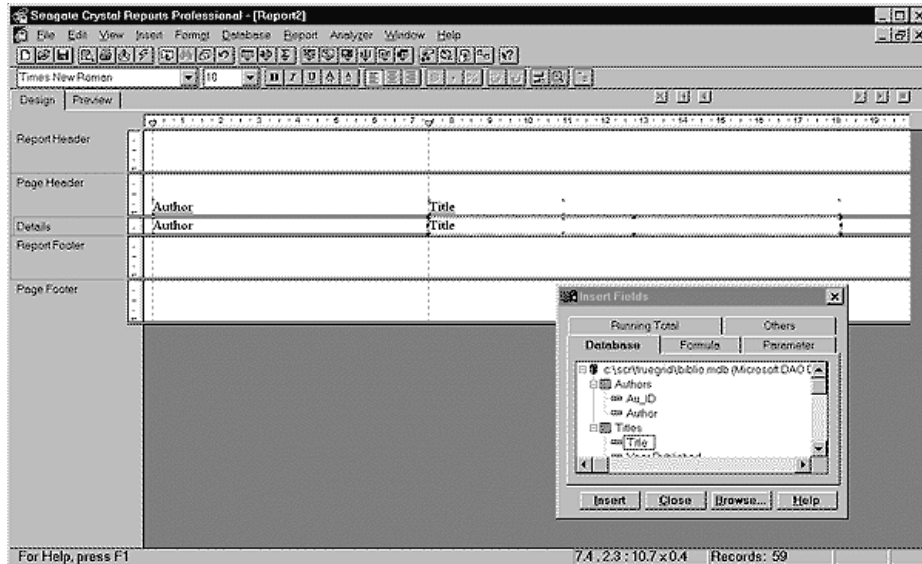


Create a new report by choosing File menu - New and the Create New Report dialog box will appear.

(see picture below)



Click on the Custom button to display the Choose Report Type and Data Type frames. Select the Custom Report and click on the Data File.



Specify the location of the BIBLIO.MDB Access database

Select which tables from the database you want. For this example we will need the authors table and the Title table.

check that the joins are correct in the Linking Export

Now we are ready to draw the report

Drag and drop the fields you want on to the report. So that it looks like the report above.

Now if you run it by selecting Print Preview, you can see what the report looks like.

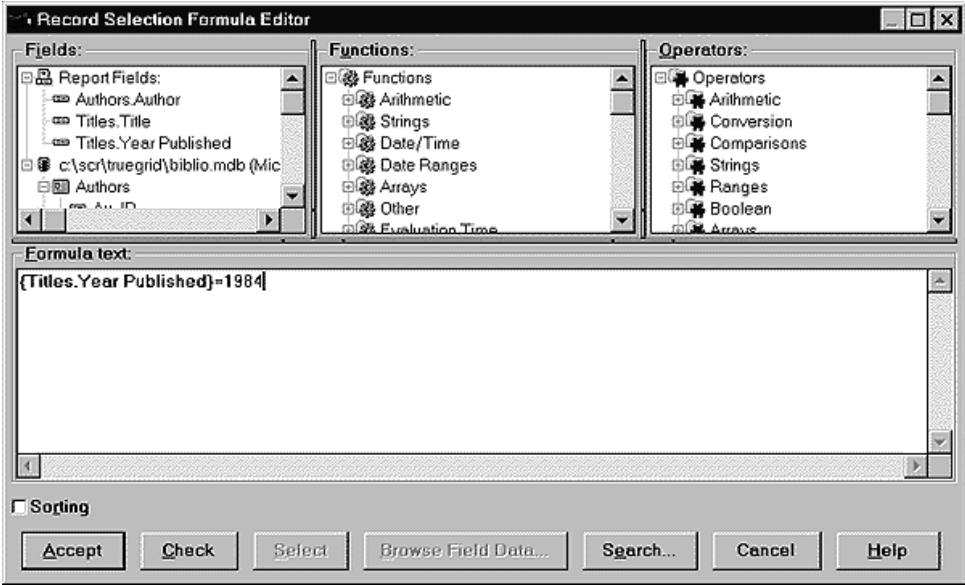
Now save the report and we have a template that now can be used in VB.



Notes

Turn off the Save Data with report option if you want the data to contain new information each time the report is used. Going to the File menu, selecting Options and clicking on the Reporting Tab sets the option

There are many options in the Crystal Reports and as this tutorial is more of an overview I will not be going into it. But one important thing to do is to get specific data. This is done by a selection formula (*i.e.*, the Where clause of a SQL statement).



Questions

1. Discuss about the Crystal Report.
2. How to use Crystal Report in the application?

2.3 Summary

- Working of VB include different stage of project application elaborated.
- Method and events is an action that can performed on object.
- Event procedures is a procedure block that contains the control's actual name an underscore (_) and the event name.

2.4 Keywords

Data environment: It is a repository in your Microsoft® Visual InterDev™ Web project for the information required in server script to connect and manipulate data in databases. It provides a standard interface for creating re-usable data-related objects and for placing them on Web pages.

Data report: Once we have gone to all trouble all developing and managing data base, it is nice to have ability to obtain printed or displayed information from data. The process of obtaining such information is known as creating a data base.



Lab Exercise

1. Give steps need to be taken to create an ActiveX Control.
2. How do I add a Data Environment to my project?
3. Give the step to add a form in application.

2.5 Self Assessment Questions

1. What action should you not do in a class's Initialize event?
 - (a) Set default values.
 - (b) Create dependent objects.
 - (c) Set an object variable to a new copy of the class.
 - (d) Open files or other resources to be used by the class.
2. You are using a program that accesses a Single Use ActiveX EXE program. After creating a second copy of an object defined by the program, what should you expect to happen?
 - (a) More memory is used.
 - (b) Memory is freed.
 - (c) An error occurs. You can only run one copy of a SingleUse program.
 - (d) Nothing.
3. You have an ActiveX Automation Server that displays forms. What must be done to have the application run unattended and support multithreads?
 - (a) You cannot have unattended Automation Servers.
 - (b) There is nothing that must be done.
 - (c) Remove the forms and code that refers to forms. Then select the Unattended Execution option.
 - (d) Select the Unattended Execution option.
4. ActiveX controls are implemented in files with the extension:
 - (a) OCX
 - (b) VBX
 - (c) VBX and OCX
 - (d) OLB and OCX
5. A sufficient condition to fire a control's Validate event is:
 - (a) The user must make a change to the control's contents and the control's CausesValidation property must be True.
 - (b) The user must make a change to a control's contents and set focus to another control on the same form whose CausesValidation property is True.
 - (c) The user must set focus to another control on the same form and the CausesValidation property of the control losing focus must be True.
 - (d) The user must set focus to a control whose CausesValidation property is True.
6. Which of the following statements will raise an event called Click with no parameters?
 - (a) RaiseEvent Click
 - (b) Call UserControl_Click
 - (c) RaiseEvent UserControl_Click
 - (d) Raise Click

7. You have created an ActiveX control. The control needs to be hidden from the user at runtime. Which of the following actions will ensure that the control is hidden at runtime?
 - (a) Setting the Visible property to False in the Initialize event of the control.
 - (b) Setting the InvisibleAtRuntime property to True.
 - (c) Call the InvisibleNow method of the ActiveX control.
 - (d) This cannot be done because ActiveX controls in Visual Basic are always visible.
8. Which of the following statements is true about ActiveX controls?
 - (a) ActiveX controls can only be used from Web pages.
 - (b) ActiveX controls cannot contain other controls.
 - (c) ActiveX controls must be visible at runtime.
 - (d) An ActiveX control can read or save its properties from a persistent location.
9. Which statement is the best description of a Property Bag object?
 - (a) The Property Bag is a collection of the ActiveX control properties.
 - (b) Property Bag is an object used to read and write properties to a persistent location.
 - (c) A Property Bag is an array of properties of an ActiveX control.
 - (d) Property Bag is not a valid object type in Visual Basic.

2.6 Review Questions

1. If You have an ActiveX EXE program with a class declared as SingleUse. What effect could this have on the memory of the computer when multiple clients access this object?
2. Have you ever created ActiveX controls? If so, what did they do?
3. Write the steps in Creating ActiveX Dll and Active Exe.
4. What is active? Why it is required in visual basic? Explain its usage.
5. Write the Steps in Creating an ActiveX Control.
6. What is the difference between ActiveX Exe and Dll?
7. Begin a new ActiveX DLL project in VB.
8. Describe how and why raising events are necessary when creating ActiveX controls.
9. Describe the facilities provided in Visual Basic that allow a UserControl to load and write its properties.
10. How does the programmer prevent an ActiveX control from being visible at runtime?
11. Describe the role of the PropertyBag object.
12. What must you do to test your ActiveX control within Internet Explorer? What must you do to test its behavior in another VB program?

13. What is the name of the collection that provides the controls being modified by a Property Page?
14. What two things must you do to an ActiveX control project so that it will provide the programmer with DataField and DataSource properties?
15. When does the GetDataMember event fire?
16. What is the difference between ActiveEXE and Standard EXE?

Answers for Self Assessment Questions

- | | | | |
|--------|--------|--------|--------|
| 1. (c) | 2. (a) | 3. (d) | 4. (a) |
| 5. (d) | 6. (a) | 7. (b) | 8. (d) |
| 9. (b) | | | |

2.7 Further Readings



G. Cornell, "Visual Basic 6" Tata McGraw-Hill, 1998.

Deitel & Deitel, T.R. Nieto, "Visual Basic 6, How to program" Prentice Hall of India, 1999.



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

Unit 3: Understanding Controls and Control Events

Contents

Objectives

Introduction

- 3.1 Design Aspects of VB Forms
 - 3.1.1 The Project Explorer Window
 - 3.1.2 The Default Layout
 - 3.1.3 Understanding the Tool Box
 - 3.1.4 Opening an Existing Visual Basic Project
 - 3.1.5 Opening a New Visual Basic File and Inserting Source Code
 - 3.1.6 Running and Viewing the Project in Detail
 - 3.1.7 Making your First *.exe!?
 - 3.1.8 Saving your Visual Basic Project
- 3.2 Element of User Interface Design Principles
 - 3.2.1 Simplicity
 - 3.2.2 Positioning of Controls
 - 3.2.3 Consistency
 - 3.2.4 Aesthetics
- 3.3 Button Properties for Reference
- 3.4 Properties of Controls
 - 3.4.1 TextBox Control
 - 3.4.2 Label
 - 3.4.3 LinkLabel
 - 3.4.4 Command Button
 - 3.4.5 CheckBox
 - 3.4.6 ListBox
 - 3.4.7 ComboBox
 - 3.4.8 Frame Control
 - 3.4.9 Option Buttons
- 3.5 Forms Properties
- 3.6 Form Load Event
- 3.7 Summary
- 3.8 Keywords
- 3.9 Self Assessment Questions
- 3.10 Review Questions
- 3.11 Further Reading

Objectives

After studying this unit, you will be able to:

- Understand the design aspects of VB forms
- Discuss the elements of user interface design principles
- Explain the button properties for reference
- Discuss the control properties
- Explain load event form.

Introduction

For our purposes, Visual Basic programs are also known as applications, solutions, or projects. Each program is saved (as several files and subfolders) in its own folder. Before starting a new program, you should use Windows Explorer to create a folder to hold the folders for your programs.

The process for invoking Visual Basic varies slightly with the edition of Visual Basic installed on the computer. To invoke Visual Basic from a computer that has Visual Basic Express installed, click the windows start button, however over all programs, and then click on Microsoft Visual Basic 2005 Express Edition. With the other editions of Visual Basic.

3.1 Design Aspects of VB Forms

Learning the ins and outs of the Development Environment before you learn visual basic is somewhat like learning for a test you must know where all the functions belong and what their purpose is. First we will start with labeling the development environment.

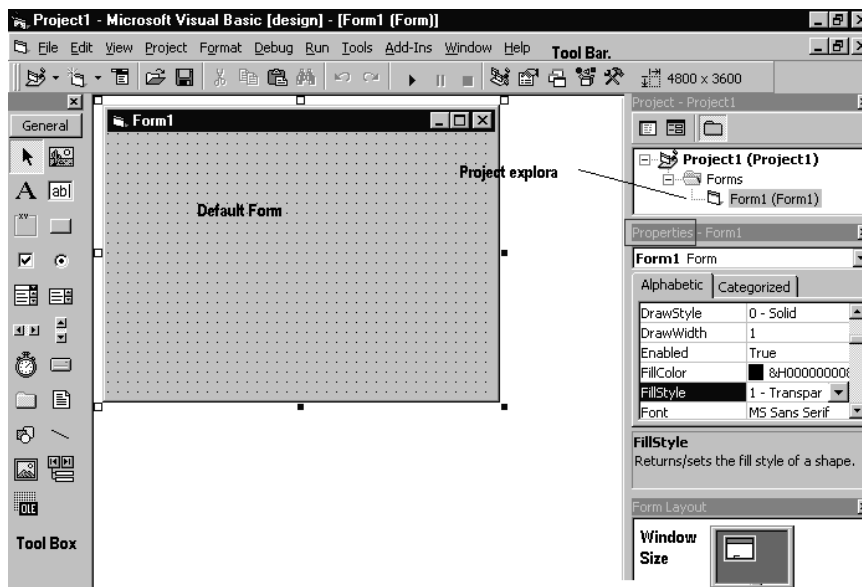


Figure 3.1: Design Aspects of VB Forms.

The above diagram shows the development environment with all the important points labelled. Many of Visual Basic functions work similar to Microsoft word e.g. the **Tool Bar** and the **Tool Box** is similar to other products on the market which work off a single click then drag the width of the object required. The **Tool Box** contains the control you placed on the form window. All of the controls that appear on the **Tool Box** controls on the above picture never runs out of controls as soon as you place one on the form another awaits you on the **Tool Box** ready to be placed as needed.

3.1.1 The Project Explorer Window

The Project explorer window gives you a tree-structured view of all the files inserted into the application. You can expand these and collapse branches of the views to get more or less detail (**Project explorer**). The project explorer window displays forms, modules or other separators which are supported by the visual basic like classes and Advanced Modules. If you want to select a form on its own simply double click on the project explorer window for a more detailed look. And it will display it where the **Default form** is located.

3.1.1.1 Properties Window

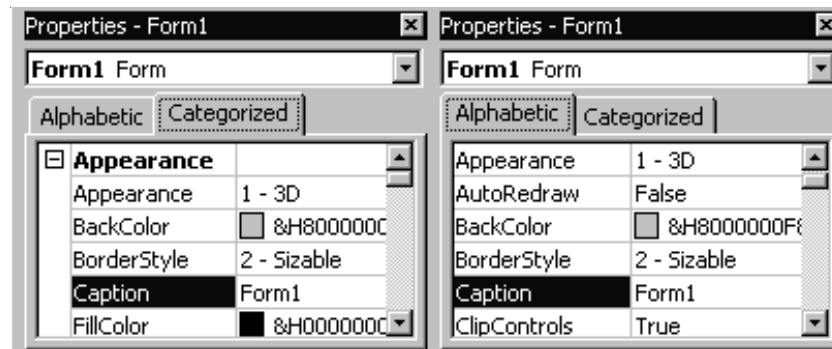


Figure 3.2: Properties Window.

Some programmers prefer the Categorized view of the properties window. By defaulting, the properties window displays its properties alphabetically (with the exception of the name value) when you click on the categorized button the window changes to left picture.

3.1.2 The Default Layout

When we start Visual Basic, we are provided with a VB project. A VB project is a collection of the following modules and files:

- The **global module**: that contains declaration and procedures.
- The **form module**: that contains the graphic elements of the VB application along with the instruction.
- The **general module**: that generally contains general-purpose instructions not pertaining to anything graphic on-screen.
- The **class module**: that contains the defining characteristics of a class, including its properties and methods.
- The **resource files**: that allows you to collect all of the texts and bitmaps for an application in one place.

On start up, Visual Basic will displays the following windows:


- The **Blank Form** window
- The **Project** window
- The **Properties** window

It also includes a **Tool Box** that consists of all the controls essential for developing a VB Application. Controls are tools such as boxes, buttons, labels and other objects draw on a form to get **input** or **display output**. They also add visual appeal.



If the Description pane is not visible, right-click on the Properties window, and then click on "Description." The Description pane describes the currently highlighted property.

3.1.3 Understanding the Tool Box

You may have noticed that when you click on different controls the **Properties Window** changes slightly this is due to different controls having different functions. Therefore more options are needed for example if you had a picture then you want to show an image. But if you wanted to open a internet connection you would have to fill in the remote host and other such settings. When you use the command () you will find that a new set of properties come up the following will provide a description and a property.

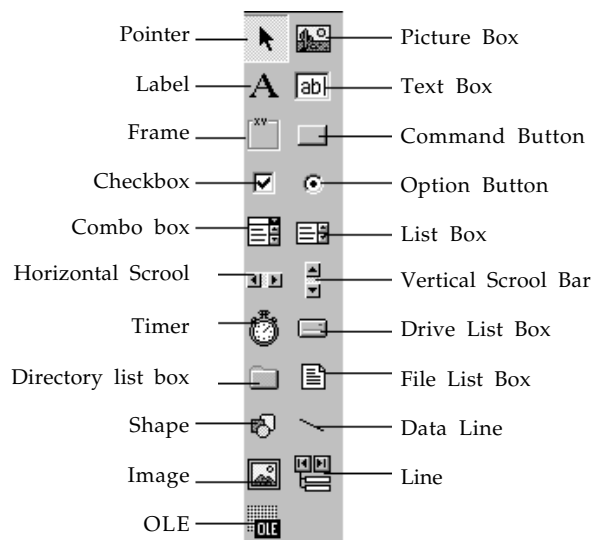


Figure 3.3: Tool Box.

3.1.4 Opening an Existing Visual Basic Project

Microsoft have included some freebies with visual basic to show its capabilities and functions. Dismantling or modifying these sample projects is a good way to understand what is happening at runtime. These files can be located at your default directory /samples/.

To open these projects choose 'Open Project' from the 'File' menu. Then double click on the samples folder to open the directory then double click on any project to load it.


3.1.5 Opening a New Visual Basic File and Inserting Source Code



From looking at the examples it time to make your own application. Choose 'New Project' from the 'File' menu. Use the blank form1 to design a simple interface for an estate agents database, have some textboxes for names and other details. Insert some controls and make it look

professional. Textboxes can be used to store there name and other details, make sure you put a picture box in for a picture of the house.

Now insert the following source code for your application. Private Sub Form_Load()
Picture1.Picture = LoadPicture("C:\Program Files\VB\Graphics\Icons\Misc\MISC42.ICO")
End Sub

3.1.6 Running and Viewing the Project in Detail

Once an application is loaded it can be run by click on the  icon from the tool bar, to pause

press  and to terminate use .

Once a project is loaded, the name of the form(s) that it contains is displayed in the project window. To view a form in design mode, select the form required by clicking with the mouse to highlight its name, then clicking on the view form button.

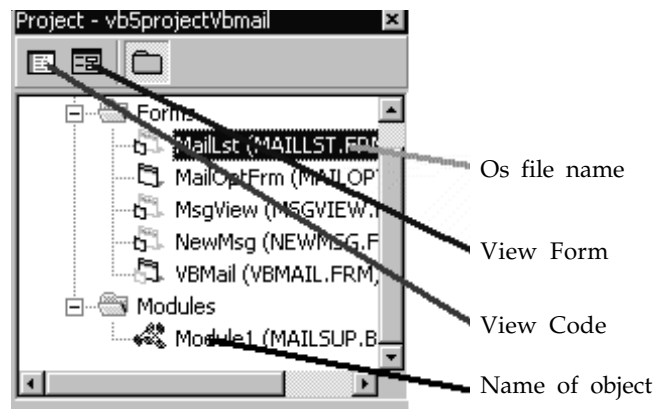


Figure 3.4: Tool Box view of Project.

In this example the project has been loaded and the maillist.frm has been selected for viewing. This Ms Mail example project uses 6 forms and 1 modules.

In Design mode, when the form is viewed, the code attached to any screen object may be inspected by double clicking on that object. The screen shots below show the interface of the Ms Mail example (.../samples/Comtool/VBMail/MaiLLST.FRM) to view the code for this form select



from the project window item.

```
Private Sub SetupOptionForm(BasePic As Control)
BasePic.Top = 0
BasePic.Left = 0
BasePic.Visible = True
BasePic.enabled = True
OKBt.Top = BasePic.Height + 120
Me.width = BasePic.Width + 120
Me.Heigh = OkBt.Top + OkBt.Height + 495
End Sub
```

3.1.7. Making Your First *.exe!?

To make an executable from a project choose 'Make project.exe' from the 'File' menu. Then click once on the Make project.exe choose a default location to store your executable, you can also change some advanced options by clicking on the **Options.** tag before saving your exe. The above image will be displayed in the comment's value type some comments company name etc. The Title tag represents the caption you will see if you press Control + Alt + Del. And the icon is the icon that will be available on the execute icon. As you can see it is quite simple to understand. All the comments, data and name appear when you click on the compiled (execute) exe and click properties.



Save the exe file name of "Alld".

3.1.8. Saving your Visual Basic Project

Save your work to disk. Use the Windows Explorer or any desktop windows to check that all files have been saved. There should be one Visual Basic Project (.VBP) file and separate Form (.FRM) and Module (.BAS) files for each form and module used in the current project.

Figure 3.5: Visual Basic Project (.VBP) File.



Did u know?

When a program is run, all the work done so far on the program is automatically saved in a temporary location with the name listed earlier in the New Project input dialog box.



If the “Create directory for solution” check box is checked, then click on the check box to uncheck it. Finally, click on the save button.

3.2 Element of User Interface Design Principles

User Interface is a very important factor when we design our application (project). User interface provides a mechanism for end users to interact with the application. End users are called target audience. Designing a good user interface which is easy to use and understand is crucial for a successful application. If you already know the target audience for whom you are developing the application then designing becomes simple as you will already be familiar with their corporate colors and likes. Target audience, for example, can be the employee’s of a firm for whom you will design the application. A well designed user interface makes it easy and simple for the target audience to understand and use. On the other hand, a poorly designed user interface will be hard to understand and use and can lead to distraction and frustration.

Good user interface is possible if it is designed keeping in mind the following four principles:

1. Simplicity
2. Positioning of Controls
3. Consistency
4. Aesthetics

The screenshot shows a web form titled "New Account Registration" within a window labeled "Register". The form contains the following elements:

- First Name: Text input field
- Last Name: Text input field
- Date of Birth: Text input field
- Sex: Radio buttons for "Male" and "Female"
- Address 1: Text input field
- Address 2: Text input field
- State: Dropdown menu with "Select State" as the placeholder text
- Buttons: "Submit" and "Reset" buttons at the bottom right

Figure 3.6: Good User Interface Design.

The above said principles explained:

3.2.1 Simplicity

Simplicity is a key factor when designing a user interface. If a user interface looks crowded with controls then learning and using that application will be hard. Simplicity means, the user interface should allow the user to complete all the required tasks by the program quickly and easily. Also, program flow and execution should be kept on mind while designing. Try to avoid use of flashy and unnecessary images that distract the user. The simpler the user interface, the more friendly and easy it will be.

3.2.2 Positioning of Controls

Positioning of controls should reflect their importance. Say, for example, if you are designing an application that has a data-entry form with textboxes, buttons, radio buttons, etc. The controls should be positioned in such a way that they are easy to located and matches the program flow. Like, a submit button should be placed at the bottom of the form so that when the user enters all the data he can click it straight away. The image above is a perfect example of positioning of controls.

3.2.3 Consistency

The user interface should have a consistent look through out the application. The key to consistency lies during the design process. Before developing an application, we need to plan and decide a consistent visual scheme for the application that will be followed throughout. Using of particular fonts for special purposes, using of colors for headings, use of images, etc. are all part of consistency.

3.2.4 Aesthetics

An application should project an inviting and pleasant user interface. The following should be considered for that.

3.2.4.1 Color

Use of color is one way to make the user interface attractive to the user. The color which you select for text and back-ground should be appealing. Care should be taken to avoid gaudy colors that are disturbing to the eye, for example, black text on a red back ground.

3.2.4.2 Fonts

The fonts which you use for text should also be selected with care. Simple, easy-to-read fonts like Verdana, Times New Roman should be used. Try to avoid bold, strikethrough text in most parts of the application. Use of bold, italics and other formatting should be limited to important text or headings.

3.2.4.3 Images

Images add visual interest to the application. Simple, plain images should be used wherever appropriate. Avoid using flashing images and images that are not necessary but are used only for show off.

3.2.4.4 Shapes and Transparency

Framework provides tools that allow us to create forms and controls with different levels of opacity. Apart from using traditional shapes like rectangles, etc, these tools also allow us to draw our own shapes which can provide some very powerful visual effects. User drawn shapes should

be used only if the application requires it and care should be taken that the shapes which are drawn do not disturb the eye.

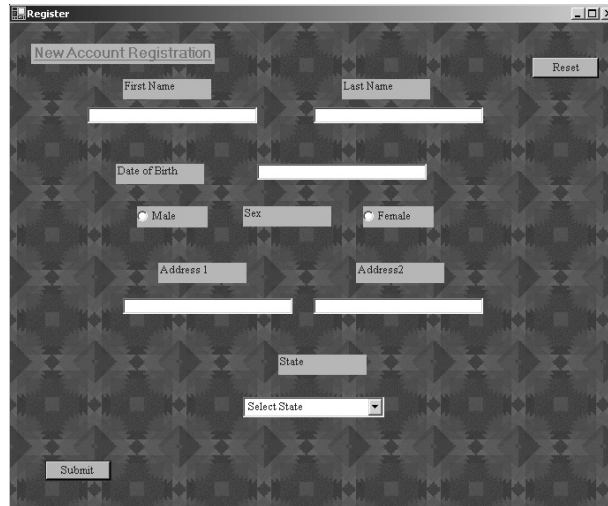


Figure 3.7: Poorly Designed User Interface.

3.3. Button Properties for Reference

Command Button and labels properties

Property	Description
Name	The name of the object so you can call it at runtime
BackColor	This specifies the command button's background color. Click the <i>BackColor's</i> palette down arrow to see a list of common Windows control colors, you must change this to the style property from 0 - standard to 1 - graphical
Cancel	Determines whether the command button gets a Click event if the user presses escape
Caption	Holds the text that appears on the command button.
Default	Determines if the command button responds to an enter keypress even if another control has the focus
Enable	Determines whether the command button is active. Often, you'll change the enable property at runtime with code to prevent the user pressing the button
Font	Produces a Font dialog box in which you can set the caption's font name, style and size
Height	Positions the height of the object—can be used for down
Left	Positions the left control—can be used for right

Mouse Pointer	If selected to an icon can change the picture of the mouse pointer over that object
Picture	Hold's the name of an icon graphic image so that it appears as a picture instead of a Button for this option to work the graphical tag must be set to 1
Style	This determines if the Command Button appears as a standard windows dialog box or a graphical image
Tab index	Specifies the order of the command button in tab order
Tab Stop	Whether the object can be tabbed to (this can be used in labels which have no other function)
Tool Tip	If the mouse is held over the object a brief description can be displayed (for
Text	example hold your mouse over one of the above pictures to see this happening
Visible	If you want the user to see the button/label select true other wise just press false
Width	Show the width of the object

3.4 Properties of Controls

A control is an object that can be drawn on to the Form to enable or enhance user interaction with the application. Some examples of these controls are TextBoxes, Buttons, Labels, Radio Buttons, etc. All these Windows Controls are based on the **Control** class, the base class for all controls. Visual Basic allows us to work with controls in two ways: at **design time** and at **runtime**. Working with controls at design time means, controls are visible to us and we can work with them by dragging and dropping them from the Toolbox and setting their properties in the properties window. Working at runtime means, controls are not visible while designing, are created and assigned properties in code and are visible only when the application is executed. There are many new controls added in Visual Basic and we will be working with some of the most popular controls in this section. You can select the controls from the menu towards the left-hand side of this page.

Notable properties of most of these Windows Controls which are based on the Control class itself are summarized in the table below. You can always find the properties of the control with which you are working by pressing **F4** on the keyboard or by selecting **View->Properties Window** from the main menu.

3.4.1 TextBox Control

Windows users should be familiar with textboxes. This control looks like a box and accepts input from the user. The TextBox is based on the TextBoxBase class which is based on the Control class. TextBoxes are used to accept input from the user or used to display text. By default we can enter up to 2048 characters in a TextBox but if the Multiline property is set to True we can enter up to 32KB of text. The image below displays a TextBox.

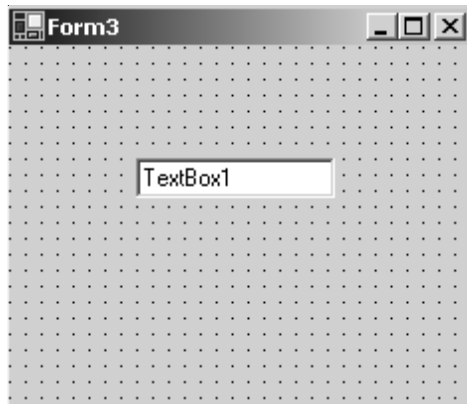


Figure 3.8: TextBox.

3.4.1.1 Some Notable Properties

Some important properties in the behavior section of the Properties Window for TextBoxes.

Enabled: Default value is True. To disable, set the property to False. **Multiline:** Setting this property to True makes the TextBox multiline which allows to accept multiple lines of text. Default value is False. **PasswordChar:** Used to set the password character. The text displayed in the TextBox will be the character set by the user. Say, if you enter *, the text that is entered in the TextBox is displayed as *. **ReadOnly:** Makes this TextBox readonly. It doesn't allow to enter any text. **Visible:** Default value is True. To hide it set the property to False.

3.4.1.2 Appearance Section Properties

TextAlign: Allows to align the text from three possible options. The default value is left and you can set the alignment of text to right or center.

Scrollbars: Allows to add a scrollbar to a Textbox. Very useful when the TextBox is multiline. You have four options with this property. Options are None, Horizontal, Vertical and Both. Depending on the size of the TextBox anyone of those can be used.

3.4.1.3 TextBox Event

The default event of the TextBox is the TextChanged Event which looks like this in code:

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As _  
System.EventArgs) Handles TextBox1.TextChanged
```

```
...  
End Sub
```

3.4.1.4 Working With TextBoxes

Lets work with some examples to understand TextBoxes.

Drag two TextBoxes (TextBox1, TextBox2) and a Button (Button1) from the toolbox.

Code to Display some text in the TextBox

We want to display some text, say, "Welcome to TextBoxes", in TextBox1 when the Button is clicked. The code looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
    TextBox1.Text = "Welcome to TextBoxes"
End Sub
```

Code to Work with PassWord Character

Set the PasswordChar property of TextBox2 to *. Setting that will make the text entered in TextBox2 to be displayed as *. We want to display what is entered in TextBox2 in TextBox1. The code for that looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
    TextBox1.Text = TextBox2.Text
End Sub
```

When you run the program and enter some text in TextBox2, text will be displayed as *. When you click the Button, the text you entered in TextBox2 will be displayed as plain text in TextBox1.

Code to Validate User Input

We can make sure that a TextBox can accept only characters or numbers which can restrict accidental operations. For example, adding two numbers of the form 27+2J cannot return anything. To avoid such kind of operations we use the KeyPress event of the TextBox. Code that allows you to enter only double digits in a TextBox looks like this:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As_
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
    If(e.KeyChar < "10" Or e.KeyChar > "100") Then
        MessageBox.Show("Enter Double Digits")
    End If
End Sub
```

Creating a TextBox in Code

```
Public Class Form1 Inherits System.Windows.Forms.Form
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
        Dim TextBox1 as New TextBox()
        TextBox1.Text="Hello Mate"
        TextBox1.Location=New Point(100,50)
        TextBox1.Size=New Size(75,23)
        Me.Controls.Add(TextBox1)
    End Sub
End Class
```



The TextBox is one of the handiest and most popular controls in a Windows application. It's a common approach to getting free-form input from the user. You can manipulate a TextBox control's contents and detect changes the user makes to the TextBox through several Properties.

3.4.2 Label

Labels are those controls that are used to display text in other parts of the application. They are based on the Control class.

Notable property of the label control is the text property which is used to set the text for the label.

3.4.2.1 Label Event

The default event of Label is the Click event which looks like this in code:

```
Private Sub Label1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)_
Handles Label1.Click
End Sub
```

3.4.2.2 Creating a Label in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles MyBase.Load Dim Label1 As New Label()
Label1.Text = "Label"
Label1.Location = New Point(135, 70)
Label1.Size = New Size(30, 30)
Me.Controls.Add(Label1)
End Sub
```

3.4.3 LinkLabel

LinkLabel is similar to a Label but they display a hyperlink. Even multiple hyperlinks can be specified in the text of the control and each hyperlink can perform a different task within the application. They are based on the Label class which is based on the Control class.

Notable properties of the LinkLabel control are the ActiveLinkColor, LinkColor and LinkVisited which are used to set the link color.

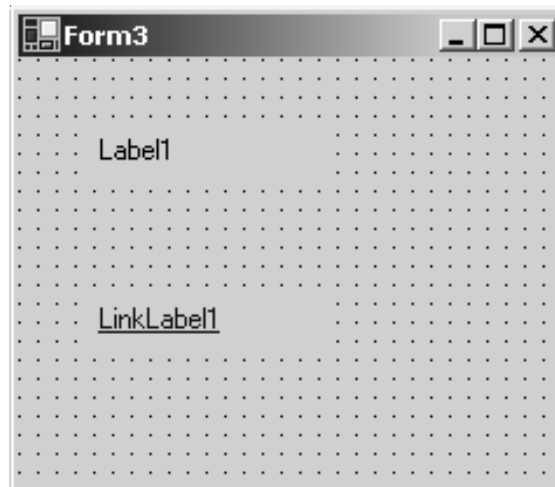


Figure 3.9: LinkLabel.

3.4.3.1 LinkLabel Event

The default event of LinkLabel is the LinkClicked event which looks like this in code:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _  
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs)_  
    Handles LinkLabel1.LinkClicked  
End Sub
```

3.4.3.2 Working with LinkLabel

Drag a LinkLabel (LinkLabel1) onto the form. When we click this LinkLabel it will take us to "www.startvbdotnet.com". The code for that looks like this:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, ByVal_  
    e As System.Windows.Forms.LinkLabelLinkClickedEventArgs)_  
    Handles LinkLabel1.LinkClicked  
    System.Diagnostics.Process.Start("www.startvbdotnet.com")  
    'using the start method of system.diagnostics.process class  
    'process class gives access to local and remote processes  
End Sub
```

3.4.3.3 Creating a LinkLabel in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_  
    Handles MyBase.Load  
    Dim LinkLabel1 As New LinkLabel()
```

```
LinkLabel1.Text = "Label"  
LinkLabel1.Location = New Point(135, 70)  
LinkLabel1.Size = New Size(30, 30)  
Me.Controls.Add(LinkLabel1)  
End Sub
```

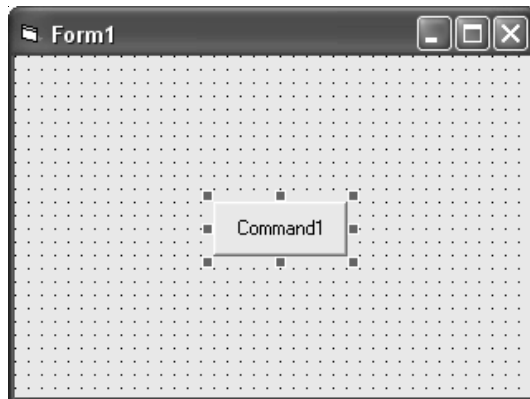


Figure 3.10: The Command Button on a Form.

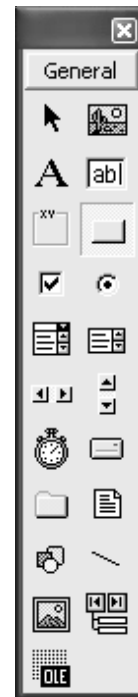


Figure 3.11: The Command Button's Icon on the Toolbox.



Did u know?

"The MultiLine property is only writable at design time, although you can find out its value at runtime. MultiLine is False by default, meaning that everything in the TextBox control will appear on a single line. If MultiLine is set to True, the TextBox will perform word-wrapping and also break a typed line after a hard return.

3.4.4 Command Button

The Command Button is used to make an action happen *e.g.* process the inputs that have been entered onto a Form and then show the output.

The naming convention for the Command Button is 'cmd'. *E.g.* cmdExit. Also, Command Button's look more professional if you set the Height property to 375, unless your creating Graphical buttons of course.

3.4.4.1 Properties of Command Button

BackColor

Sets the color of the Command Button's background. This property only takes effect when the Style property is set to 1 - Graphical.

Cancel

If this property is set to True, pressing the Escape key at runtime will activate the Command Button's Click event. Only one Command Button on the Form can have this property set to True.

Caption

Sets the text to be displayed on the Command Button.

If possible keep this as short as possible, so you don't have to resize the Command Button. You can always use the ToolTipText property if you want to give the user further information.

You can use mnemonics *e.g.* putting the '&' symbol somewhere in the Caption will cause the following character to be underlined at runtime. When the user presses the Alt+[Underlined Letter] key combination, the Command Button's Click event will be called.

Default

Works in a similar way to the cancel property, but with the Enter key. The Command Button, which has this property set to True, will also appear with a darker border.

Disabled Picture

Sets the picture to be displayed when the enabled property is set to False. The Style property must be set to 1-Graphical.

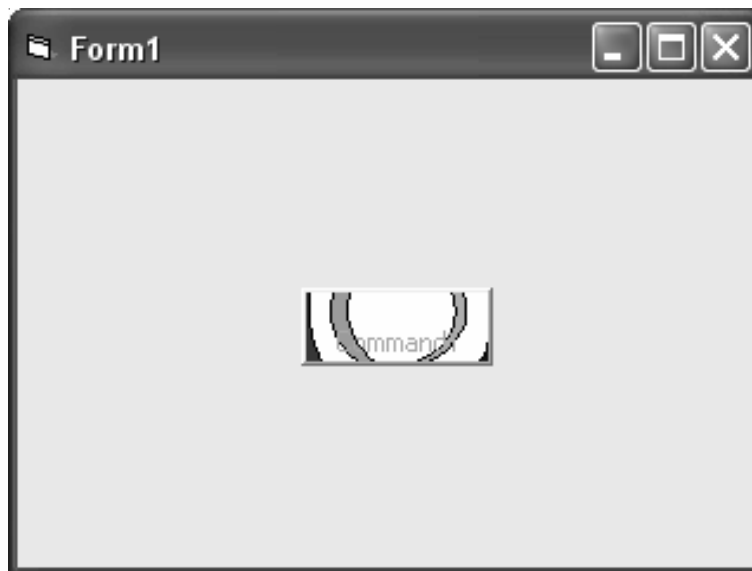


Figure 3.12: Using the Disabled Picture Property.

If the Command Button was big enough, the caption would appear under the picture instead of over it.

Down Picture

Sets the picture to be displayed while the button is pressed down *i.e.*, when the user holds the mouse button or the space key on it. The Style property must be set to 1 - Graphical.

Enabled

When this property is set to True, the user is not able to interact with the Command Button in any way *e.g.* it cannot be clicked on or have focus. The caption will appear greyed out.

Font

In design time, this brings up the Font dialog box to allow you to choose the font name, style, size and whether the Strikeout and Underline attributes are set. When writing code to set the font of the Command Button, Font acts as an object, which has its own properties:

- Bold
- Charset
- Italic
- Name
- Size
- Strikethrough
- Underline
- Weight



Example: `Command1.Font.Bold = True`

`Command1.Font.Italic = True`

`Command1.Font.Underline = True` Alternatively, you can use the `FontBold` (Boolean), `FontItalic` (Boolean), `FontName` (String), `FontSize` (Single), `FontStrikethru` (Boolean) and `FontUnderline` (Boolean) properties, which are only available at runtime.

Height

Sets the height of the Command Button.

Index

Leave blank unless the Command Button is to be part of a control array.

Left

Sets the X coordinate of the Command Button.

MaskColor

This property only comes into effect when the following conditions are met - the Style property is set to 1 - Graphical, the UseMaskColor property is set to True and a background picture is being

displayed on the Command Button. The color that is set in the MaskColor property will become the transparent color in the picture. E.g. if white is used in the picture, setting the MaskColor to white will mean that white changes the background color of the Command Button (*i.e.*, changes to being transparent).

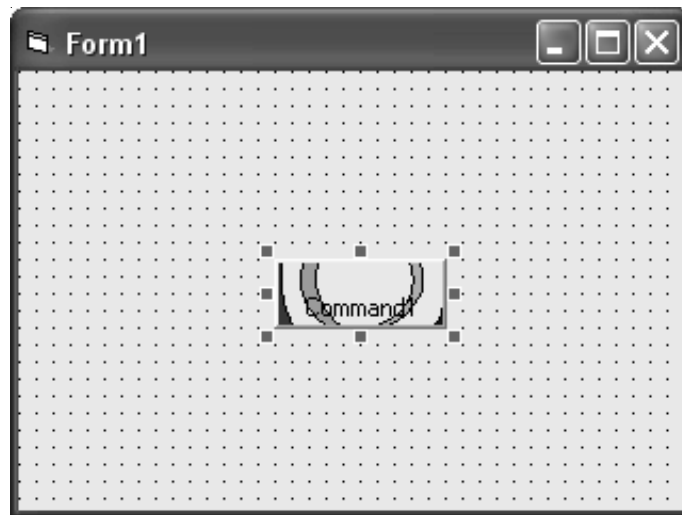


Figure 3.13: The MaskColor has been set to white, which means that the white areas in the picture have become transparent.

You should not use a system color for the MaskColor property because system colors are changeable by the user. This means that the Command Button would end up with no color being made transparent, which would not be the desired effect.

Mouse Icon

When the user hover's the mouse pointer over the Command Button, the image set for this property will become the mouse icon. This property will only work if the Mouse Pointer property is set to 99 - Custom.

Mouse Pointer

Sets the type of mouse pointer to be used when the user hovers the mouse over the Command Button. The possible choices are:

Table 3.1: Mouse Pointer Choices

	Value	Name	Constant
0		Default	vbDefault
1		Arrow	vbArrow
2		Cross	vbCrosshair

3	I-Beam	vbIbeam
4	Icon	vbIconPointer
5	Size	vbSizePointer
6	Size NE SW	vbSizeNESW
7	Size NS	vbSizeNS
8	Size NW SE	vbSizeNWSE
9	Size W E	vbSizeWE
10	Up Arrow	vbUpArrow
11	Hourglass	vbHourglass
12	No Drop	vbNoDrop
13	Arrow and hourglass	vbArrowHourglass
14	Arrow and Question	vbArrowQuestion
15	Size All	vbSizeAll
99	Custom	vbCustom

Picture

Sets the picture, which is displayed in the background of the Command Button. If the picture is smaller than the Command Button, it will appear in the centre and the Command Button's caption will appear underneath. The Style property needs to be set to 1 - Graphical for this property to work.

Style

This needs to be set to 1 - Graphical if you want to use the graphical properties of the Command Button (apart from Fonts).

TabIndex

Sets the order in which the controls will get focus when the tab key is pressed. The control that has 0 as its TabIndex will have focus when the Form is loaded.

TabStop

If you set this to False, the Command Button will not get focus when the tab key is pressed.

Tag

This property can be set to a string if you need to store any extra information related to the Command Button.

ToolTipText

When this property is set, the text will popup over the Command Button when the user hovers the mouse pointer over it. This can be useful if you cannot fit all the text you need to in the Caption.

Top

Sets the Y coordinate of the Command Button.

Use MaskColor

If set to True, the MaskColor property will be used as the transparent color in the picture used on the Command Button. See the MaskColor property for information.

Visible

Determines whether the Command Button appears on the Form. I.e. if set to False, the Command Button will be unavailable to the user - even by trying to set focus to it with the tab key.

Width

Sets the width of the Command Button.

3.4.4.2 Methods

Move

Left As Single, [Top], [Width], [Height]

This method changes the position and optionally the size of the Command Button. The following example will move the Command Button to the top-left of the Form and makes it into a square:

```
Command1.Move 0, 0, 375, 375 SetFocus
```

Sets the focus to the Command Button. This means that pressing the Enter key will activate the Click event, even if another Command Button has it's Default property set to True.

3.4.4.3 Events

Click

This is probably the most important event for the Command Button. It occurs when the user clicks on the Command Button, or activates it in another way *e.g.* with the Enter key, Space key, Alt+[Underlined Letter] or Escape key depending on the properties of the Command Button.

```
Private Sub Command1_Click()  
    Text3.Text = CStr(Val(Text1.Text) + Val(Text2.Text))  
End Sub
```

GotFocus

Occurs when the Command Button gets focus either from the Tab key, when it is clicked on by the user or by using the SetFocus method.

KeyDown andKeyUp

KeyCode As Integer, Shift As Integer

When the user presses a keyboard key down or releases a key, these events occur.

```
Private Sub Command1_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeyBack Then
MsgBox "You pressed the backspace key!"
End If
End Sub
```

The Shift argument contains the value of the Shift, Control and Alt keys. This is useful for performing a different action when one of these keys is pressed in combination with another *e.g.* pressing 'A' would be different to pressing 'Ctrl+A'.

Constant	Value	Description
vbShiftMask	1	Shift key
vbCtrlMask	2	Ctrl key
vbAltMask	4	Alt key

If a combination of these keys are pressed, the Shift argument will contain the sum of all keys pressed *e.g.* Ctrl+Alt would be 6.

```
Private Sub Command1_KeyDown(KeyCode As Integer, Shift As Integer)
If Shift = vbCtrlMask + vbShiftMask Then
If KeyCode = vbKeyA Then
MsgBox "You pressed the Ctrl+Shift+A key combination!"
End If
End If
End Sub
```

KeyPress

KeyAscii As Integer

This event works differently to the KeyDown and KeyUp events in that it detects the actual character (as the ANSI keycode) that is pressed rather than the keyboard key. For example:

Key Pressed	Result
s	115
Shift	Nothing
Space	32

You can use the Chr function to convert the code into a letter *e.g.*, Chr(KeyAscii). If KeyAscii is set to 0 in this event, this makes the program act as if the key was never pressed.

LostFocus

Occurs when the Command Button loses focus.

Mouse Down, MouseUp

Button As Integer, Shift As Integer, X As Single, Y As Single

This event occurs when a mouse button is pressed down or released over a Command Button.

```
Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X As Single,
Y As Single)
If Button = vbMiddleButton Then
MsgBox "The middle button was pressed on the Command Button"
End If
End Sub
```

The Button argument returns one of the following possible values:

Constant	Value	Description
vbLeftButton	1	Left mouse button
vbRightButton	2	Right mouse button
vbMiddleButton	4	Middle mouse button

The Shift argument works the in same the same way as in the KeyDown and KeyUp events. The X and Y arguments return the position of the cursor in relation to the Command Button *i.e.*, the coordinates of the very top-left of the Command Button are 0, 0.

The MouseDown event occurs before the Click event and the MouseUp event occurs after the Click event.

3.4.4.4 Other Things to Try

Colored Text

It is not possible to change the color of the text of the Command Button's text as there is no Fore Color property. The only way to do it is to create a picture of colored text and use it as the Picture property. Remember to set the following properties:

- Caption: "" (no text)
- MaskColor: the background color of the picture if its not already transparent
- Picture: the picture of the colored text
- Style: 1 - Graphical
- UseMaskColor: True (if necessary)

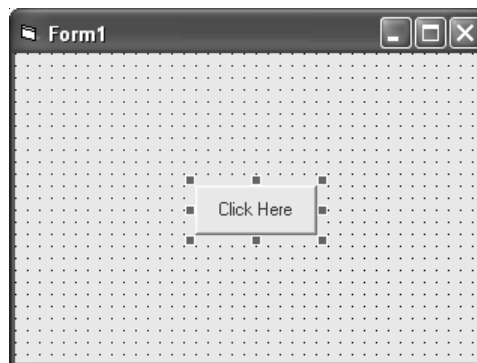


Figure 3.14: A Command Button with Colored Text.



Did u know?

Only one Command Button on a form can have its Default property set to True at a time. If you set one Command Button to be the default, all other buttons will have their Default property set to False. The same rule applies for the Cancel property.

3.4.5 CheckBox

CheckBoxes are those controls which gives us an option to select, say, Yes/No or True/False. A CheckBox is clicked to select and clicked again to deselect some option. When a checkbox is selected a check (a tick mark) appears indicating a selection. The CheckBox control is based on the TextBoxBase class which is based on the Control class. Below is the Figure 3.15 of a Checkbox.

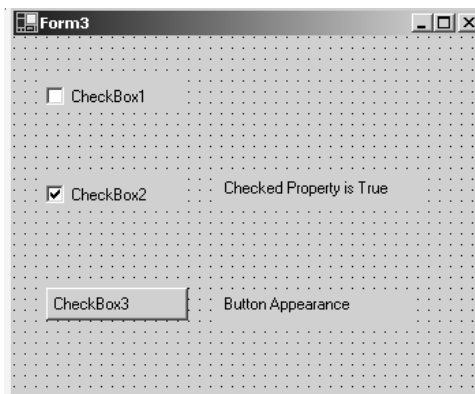


Figure 3.15: CheckBox.

3.4.5.1 Notable Properties

Important properties of the CheckBox in the Appearance section of the properties window are:

Appearance: Default value is Normal. Set the value to Button if you want the CheckBox to be displayed as a Button.

BackgroundImage: Used to set a background image for the CheckBox.

CheckAlign: Used to set the alignment for the CheckBox from a predefined list.

Checked: Default value is False, set it to True if you want the CheckBox to be displayed as checked.

CheckState: Default value is Unchecked. Set it to True if you want a check to appear. When set to Indeterminate it displays a check in gray background.

FlatStyle: Default value is Standard. Select the value from a predefined list to set the style of the checkbox.

Important property in the Behavior section of the properties window is the Three State property which is set to False by default. Set it to True to specify if the Checkbox can allow three check states than two.

3.4.5.2 CheckBox Event

The default event of the CheckBox is the CheckedChange event which looks like this in code:

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged  
End Sub
```

3.4.5.3 Working with CheckBoxes

Lets work with an example. Drag a CheckBox (CheckBox1), TextBox (TextBox1) and a Button (Button1) from the Toolbox.

```
Code to display some text when the Checkbox is checked  
Private Sub CheckBox1_CheckedChanged(ByVal sender As  
    System.Object, _  
    ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged  
    TextBox1.Text = "CheckBox Checked"  
End Sub
```

Code to check a CheckBox's state

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_  
    System.EventArgs) Handles Button1.Click  
    If CheckBox1.Checked = True Then  
        TextBox1.Text = "Checked"  
    Else  
        TextBox1.Text = "UnChecked"  
    End If  
End Sub
```

Creating a CheckBox in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_  
System.EventArgs) Handles MyBase.Load  
Dim CheckBox1 As New CheckBox()  
CheckBox1.Text = "Checkbox1"  
CheckBox1.Location = New Point(100, 50)  
CheckBox1.Size = New Size(95, 45)  
Me.Controls.Add(CheckBox1)  
End Sub
```

3.4.6 ListBox

The ListBox control displays a list of items from which we can make a selection. We can select one or more than one of the items from the list. The ListBox control is based on the ListControl class which is based on the Control class. The Figure 3.16 displays a ListBox.

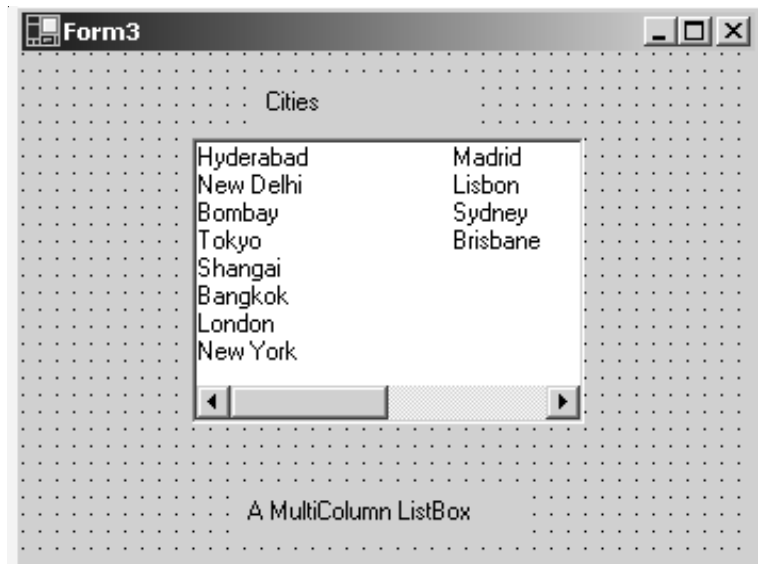


Figure 3.16: ListBox.

3.4.6.1 Notable Properties of the ListBox

3.4.6.1.1 In the Behavior Section

Horizontal Scrollbar: Displays a horizontal scrollbar to the ListBox. Works when the ListBox has Multiple Columns.

Multi Column: The default value is set to False. Set it to True if you want the list box to display multiple columns.

Scroll AlwaysVisible: Default value is set to False. Setting it to True will display both Vertical and Horizontal scrollbar always.

SelectionMode: Default value is set to one. Select option None if you do not any item to be selected. Select it to MultiSimple if you want multiple items to be selected. Setting it to MultiExtended allows you to select multiple items with the help of Shift, Control and arrow keys on the keyboard.

Sorted: Default value is set to False. Set it to True if you want the items displayed in the ListBox to be sorted by alphabetical order.

3.4.6.1.2 In the Data Section

Notable property in the Data section of the Properties window is the Items property. The Items property allows us to add the items we want to be displayed in the list box. Doing so is simple, click on the ellipses to open the String Collection Editor window and start entering what you want to be displayed in the ListBox. After entering the items click OK and doing that adds all the items to the ListBox.

3.4.6.2 ListBox Event

The default event of ListBox is the SelectedIndexChanged which looks like this in code:

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles
ListBox1.SelectedIndexChanged
End Sub
```

3.4.6.3 Working with ListBoxes

Drag a TextBox and a ListBox control to the form and add some items to the ListBox with its items property.

Referring to Items in the ListBox

Items in a ListBox are referred by index. When items are added to the ListBox they are assigned an index. The first item in the ListBox always has an index of 0 the next 1 and so on.

Code to display the index of an item

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
TextBox1.Text = ListBox1.SelectedIndex
'using the selected index property of the list box to select the index
End Sub
```

When you run the code and select an item from the ListBox, its index is displayed in the textbox.

Counting the number of Items in a ListBox

Add a Button to the form and place the following code in its click event.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
TextBox1.Text = ListBox1.Items.Count
'counting the number of items in the ListBox with the Items.Count
End Sub
```

When you run the code and click the Button it will display the number of items available in the ListBox.

Code to display the item selected from ListBox in a TextBox

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged  
    TextBox1.Text = ListBox1.SelectedItem  
    'using the selected item property  
End Sub
```

When you run the code and click an item in the ListBox that item will be displayed in the TextBox.

Code to Remove items from a ListBox You can remove all items or one particular item from the list box.

Code to remove a particular item

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _  
    As System.EventArgs) Handles Button1.Click  
    ListBox1.Items.RemoveAt(4)  
    'removing an item by specifying it's index  
End Sub
```

Code to Remove all items

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    ListBox1.Items.Clear()  
    'using the clear method to clear the list box  
End Sub
```

3.4.7 ComboBox

ComboBox is a combination of a TextBox and a ListBox. The ComboBox displays an editing field (TextBox) combined with a ListBox allowing us to select from the list or to enter new text. ComboBox displays data in a drop-down style format. The ComboBox class is derived from the ListBox class. Figure 3.17 is the image of a ComboBox.

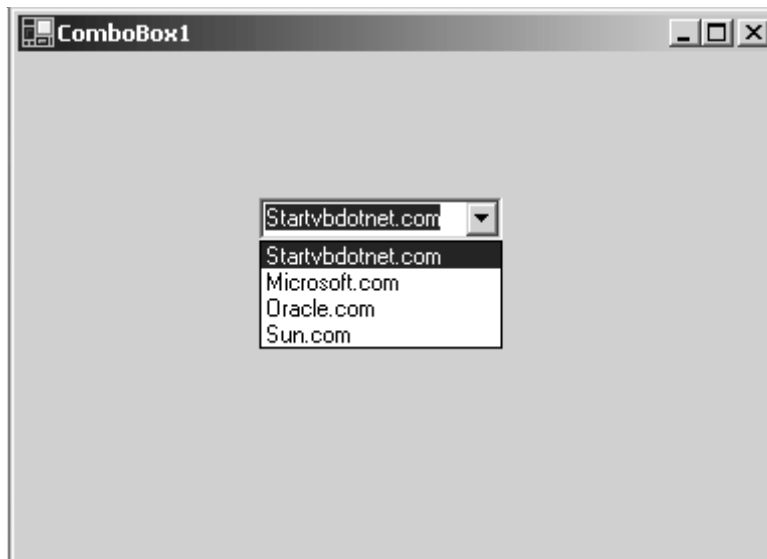


Figure 3.17: ComboBox.

3.4.7.1 Notable Properties of the ComboBox

The DropDownStyle property in the Appearance section of the properties window allows us to set the look of the ComboBox. The default value is set to DropDown which means that the ComboBox displays the Text set by its Text property in the Textbox and displays its items in the DropDownList below. Setting it to simple makes the ComboBox to be displayed with a TextBox and the list box which doesn't drop down. Setting it to DropDownList makes the ComboBox to make selection only from the drop down list and restricts you from entering any text in the textbox.

We can sort the ComboBox with its Sorted property which is set to False by Default.

We can add items to the ComboBox with its Items property.

3.4.7.2 ComboBox Event

The default event of ComboBox is SelectedIndexChanged which looks like this in code:

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged
End Sub
```

3.4.7.3 Working with ComboBoxes

Drag a ComboBox and a TextBox control onto the form. To display the selection made in the ComboBox in the TextBox the code looks like this:

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged
    TextBox1.Text = ComboBox1.SelectedItem
    'selecting the item from the ComboBox with selected item property
End Sub
```

Removing items from a ComboBox

You can remove all items or one particular item from the list box part of the ComboBox. Code to remove a particular item by its Indexnumber looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles Button1.Click
    ComboBox1.Items.RemoveAt(4)
    'removing an item by specifying its index
End Sub
```

Code to remove all items from the ComboBox

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles Button1.Click
    ComboBox1.Items.Clear()
    'using the clear method to clear the list box
End Sub
```

3.4.8 Frame Control

The Visual Basic 6.0 Frame control is replaced by two controls in Visual Basic 2005: the Group Box control and the Panel control.

3.4.8.1 Conceptual Differences

In Visual Basic 6.0, the Frame control is used as a container for grouping controls. In Visual Basic 2005, the Frame control is replaced by either the GroupBox control or the Panel control.

The GroupBox control is the equivalent of a Frame control with a BorderStyle property of 1 – Fixed Single. It has a visible border and optionally a caption.

The Panel control is the equivalent of a Frame control with a BorderStyle property of 0 – None. It has no border or caption.

In addition, there are numerous conceptual differences that apply to all controls, including differences in data binding, font handling, drag and drop, Help support and more.

3.4.8.2 Frame Control Property, Method, and Event Equivalencies

The following tables list Visual Basic 6.0 properties, methods, and events, along with their Visual Basic 2005 equivalents. Those properties, methods, and events that have the same name and

behavior are not listed. Where applicable, constants are indented beneath the property or method. All Visual Basic 2005 enumerations map to the System.Windows.Forms namespace unless otherwise noted.

Links are provided as necessary to topics explaining behavior differences. Where there is no direct equivalent in Visual Basic 2005, links are provided to topics that present alternatives.

3.4.8.3 Frame Properties

Visual Basic 6.0	Visual Basic 2005 Equivalent
Appearance	FlatStyle (GroupBox control only)
BackColor	BackColor
BorderStyle	BorderStyle (GroupBox control only)
Caption	Text (GroupBox control only)
ClipControls	New implementation.
Container	Parent
DragIcon	New implementation
DragMode	
Font	Font
Font Bold	
FontItalic	
FontName	
FontSize	
FontStrikethrough	
FontUnderline	
ForeColor	ForeColor
Height	Height, Size
HelpContextID	New implementation
HWND	Handle
Index	New implementation

Left	Left
MouseIcon	New implementation
MousePointer	Cursor For a list of constants, see MousePointer for Visual Basic 6.0 Users.
OLEDropMode	New implementation.
Parent	FindForm method
RightToLeft	RightToLeft
ToolTipText	ToolTip component
Top	Top
WhatsThisHelpID	New implementation
Width	Width, Size

3.4.8.4 Frame Methods

Visual Basic 6.0	Visual Basic 2005 Equivalent
Drag	New implementation
Move	SetBounds
OLEDrag	New implementation
ShowWhatsThis	New implementation
ZOrder	BringToFront or SendToBack function

3.4.8.5 Frame Events

Visual Basic 6.0	Visual Basic 2005 Equivalent
Click	Click (Panel only; GroupBox has no equivalent)
DblClick	DoubleClick (Panel only; GroupBox has no equivalent)

3.4.8.6 GroupBox Class

Represents a Windows control that displays a frame around a group of controls with an optional caption.

Namespace: **System.Windows.Forms**

Assembly: System.Windows.Forms (in system.windows.forms.dll)

Syntax

VB	C#	C++	F#	JScript
<pre> 'Declaration <ClassInterfaceAttribute(ClassInterfaceType.AutoDispatch)> _ <ComVisibleAttribute(True)> _ Public Class GroupBox Inherits Control 'Usage Dim instance As GroupBox </pre>				
<div style="text-align: right;">Copy</div>				

Copy

Remarks: The **GroupBox** displays a frame around a group of controls with or without a caption. Use a **GroupBox** to logically group a collection of controls on a form. The group box is a container control that can be used to define groups of controls.

The typical use for a group box is to contain a logical group of **RadioButton** controls. If you have two group boxes, each of which contain several option buttons (also known as radio buttons), each group of buttons is mutually exclusive, setting one option value per group.

You can add controls to the **GroupBox** by using the **Add** method of the **Controls** property.

The **GroupBox** cannot display a scroll bar. If you need a control similar to a **GroupBox** that can contain a scroll bar, see the **Panel** control.

3.4.9 Option Buttons

3.4.9.1 Changing the Background Color

Changing the background color gets used mostly by freeware, or the type of programs you generate for use with banners or adverts, anyway it might come in useful sometime. This example shows an ordinary picture of a smiley face then I put in some nice background colors to make it stand out more.



Figure 3.18: Background Changing.

```
Private Sub Form_Load()  
    BackColor = QBColor(Rnd * 15)  
    ForeColor = QBColor(Rnd * 10)  
    Picture1.BackColor = QBColor(Rnd * 15)  
    Picture1.ForeColor = QBColor(Rnd * 10)  
End Sub
```

3.4.9.2 PictureBox Control

PictureBoxes are used to display images on them. The images displayed can be anything varying from Bitmap, JPEG, GIF, PNG or any other image format files. The PictureBox control is based on the Control class.

Notable property of the PictureBox Control in the Appearance section of the properties window is the Image property which allows to add the image to be displayed on the PictureBox.

Adding Images to PictureBox

Images can be added to the PictureBox with the Image property from the Properties window or by following lines of code.

```
Private Sub Button1_Click(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles Button1.Click  
    PictureBox1.Image = Image.FromFile("C:\sample.gif")  
    'loading the image into the picturebox using the FromFile method of  
    the image class'  
    assuming a GIF image named sample in C: drive  
End Sub
```



Figure 3.19: PictureBox.

3.5 Forms Properties

Below are the properties of a Windows Form. Properties displayed below are categorized as seen in the properties window.

Appearance Properties	Description
BackColor	Gets/Sets the background color for the form
BackgroundImage	Get/Sets the background image in the form
Cursor	Gets/Sets the cursor to be displayed when the user moves the mouseover the form
Font	Gets/Sets the font for the form
Fore Color	Gets/Sets the foreground color of the form
Form BorderStyle	Gets/Sets the border style of the form
RightToLeft	Gets/Sets the value indicating if the alignment of the control's elements is reversed to support right-to-left fonts
Text	Gets/Sets the text associated with this form
Behavior Properties	Description
Allow Drop	Indicates if the form can accept data that the user drags and drops into it
Context Menu	Gets/Sets the shortcut menu for the form

Enabled	Gets/Sets a value indicating if the form is enabled
ImeMode	Gets/Sets the state of an Input Method Editor
Data Properties	Description
Data Bindings	Gets the data bindings for a control
Tag	Gets/Sets an object that contains data about a control
Design Properties	Description
Name	Gets/Sets name for the form
Draw Grid	Indicates whether or not to draw the positioning grid
GridSize	Determines the size of the positioning grid
Locked	Gets/Sets whether the form is locked
Snap To Grid	Indicates if the controls should snap to the positioning grid
Layout Properties	Description
Auto Scale	Indicates if the form adjusts its size to fit the height of the font used on the form and scales its controls
Auto Scroll	Indicates if the form implements autoscrolling
AutoScrollMargin	The margin around controls during auto scroll
AutoScrollMinSize	The minimum logical size for the auto scroll region
DockPadding	Determines the size of the border for docked controls
Location	Gets/Sets the co-ordinates of the upper-left corner of the form
MaximumSize	The maximum size the form can be resized to
MinimumSize	The maximum size the form can be resized to
Size	Gets/Sets size of the form in pixels
StartPosition	Gets/Sets the starting position of the form at run time
WindowState	gets/Sets the form's window state
Misc Properties	Description
AcceptButton	Gets/Sets the button on the form that is pressed when the user uses the enter key
CancelButton	Indicates the button control that is pressed when the user presses the ESC key
KeyPreview	Determines whether keyboard controls on the form are registered with the form
Language	Indicates the current localizable language

Localizable	Determines if localizable code will be generated for this object
Window Style Properties	Description
ControlBox	Gets/Sets a value indicating if a control box is displayed
HelpButton	Determines whether a form has a help button on the caption bar
Icon	Gets/Sets the icon for the form
IsMdiContainer	Gets/Sets a value indicating if the form is a container for MDI child forms
MaximizeBox	Gets/Sets a value indicating if the maximize button is displayed in the caption bar of the form
Menu	Gets/Sets the MainMenu that is displayed in the form
MinimizeBox	Gets/Sets a value indicating if the maximize button is displayed in the caption bar of the form
Opacity	Determines how opaque or transparent the form is
ShowInTaskbar	Gets/Sets a value indicating if the form is displayed in the Windows taskbar
SizeGripStyle	Determines when the size grip will be displayed for the form
TopMost	Gets/Sets a value indicating if the form should be displayed as the topmost form of the application
Transparencykey	A color which will appear transparent when painted on the form

3.6 Form Load Event

The default event of a form is the load event which looks like this in code:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load
End Sub
```

You can write code in the load event of the form just like you write for all other controls.



Example: You can run the Form by selecting Debug->Start from the main menu or by pressing F5 on the keyboard. When you run a blank form with no controls on it, nothing is displayed. It looks like the image below.



Figure 3.20: Blank Form.

Now, add a TextBox and a Button to the form from the toolbox. The toolbox can be selected from **View->ToolBox** on the main menu or by holding **Ctrl+Alt+X** on the keyboard. Once adding the TextBox and Button is done, run the form. The output window displays a TextBox and a Button. When you click the Button nothing happens. We need to write an event for the Button stating something should happen when you click it. To do that get back to design view and double-click on the Button. Doing that opens an event handler for the Button where you specify what should happen when you click the button. That looks like this in code.

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    #Region " Windows Form Designer generated code "
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
        Handles MyBase.Load
    End Sub
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

Place the code `TextBox1.Text="Welcome to Forms"` in the Click event of the Button and run the application. When you click the Button the output "Welcome to Forms" is displayed in the TextBox.

Alternatively, you can also use the **MsgBox** or **MessageBox** functions to display text when you click on the Button. To do that place a Button on the form and double-click on that to open its event. Write this line of code, `MsgBox("Welcome to Forms")` or `MessageBox.Show("Welcome to Forms")`.

It looks like this in code.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles Button1.Click
MsgBox("Welcome to Forms")
End Sub
or
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles Button1.Click
MessageBox.Show("Welcome to Forms")
End Sub
```

When you run the form and click the Button, a small message box displays, "Welcome to Forms". The image below displays that.



Figure 3.21



Caution

Make sure that the new filename has the extension ".vb".



Case Study

How to use ComboBox in visual basic?

The ComboBox is called that because it's a combination of a ListBox and a TextBox. It has the advantage over the ListBox of not taking up space until it is actually used which means that it makes it easier to position on the Form.

But the combo has the disadvantage, sort of, that the user can enter his own information, in addition to what is in the list. This may make data validation harder because the choices are not limited. When you want to force the user to make a choice only among the specified items, use a ListBox, even if it is a bit more awkward. If the user is allowed to override the choices, uses a ComboBox.

As in the ListBox, use the Text property to get the information input.

Label3.Caption = cbo_position.Text

Option Explicit

```

Private Sub Form_Load()
    'When the form loads, the first thing
    'we do is to assign the names to the
    'list of teams
    lst_team.AddItem "Giants"
    lst_team.AddItem "Redskins"
    lst_team.AddItem "Cowboys"
    lst_team.AddItem "Bears"
    lst_team.AddItem "Jets"
    'Next, Load the combo for Position
    cbo_position.AddItem "Guard"
    cbo_position.AddItem "Tackle"
    cbo_position.AddItem "Quarterback"
    cbo_position.AddItem "Receiver"
    cbo_position.AddItem "Centre"
    cbo_position.AddItem "Running back"
End Sub

Private Sub cb_go_Click()
    Label13.Caption = cbo_position.Text _
        & ", " & lst_team.Text
End Sub

```

As you can see, it is fairly simple to load the ListBox and the ComboBox during the Form_Load event. The only other detail to note is that the order in which the items appear in the Combo is not the same as the order in which the items were added. That is intentional - it is done with the **Sorted** property for the ComboBox. It can also be done for the ListBox.

Questions

1. Create a form with two buttons, run the program, and click on each button. Do you notice anything different about a button after it has been clicked?
2. While a program is running, a control is said to lose focus when the focus moves from that control to another control. In what three ways can the user cause a control to lose focus?

3.7 Summary

- Discuss design aspects of VB forms develop environment with all the important points labeled.
- User interface provides mechanism for end users to interact with the application, end user are called the target audience.
- Control is an object that can be drawn on to the form to enable or enhance user interaction with the application.
- Form load event in which you can load and display the event into a particular form.

3.8 Keywords

CheckBoxes: The *CheckBoxes* are those controls which gives us an option to select, say, Yes/No or True/False.

ComboBox: The *ComboBox* displays an editing field (TextBox) combined with a ListBox allowing us to select from the list or to enter new text. It displays data in a drop-down style format.

Command Button: The *Command Button* is used to make an action happen e.g. process the inputs that have been entered onto a Form and then show the output.

Control: A *control* is an object that can be drawn on to the Form to enable or enhance user interaction with the application.

GroupBox: The *GroupBox* displays a frame around a group of controls with or without a caption.

Labels: *Labels* are those controls that are used to display text in other parts of the application.

ListBox: The *ListBox* control displays a list of items from which we can make a selection.

LinkLabel: A *LinkLabel* is similar to a Label but they display a hyperlink.

PictureBoxes: The *PictureBoxes* are used to display images on them.

Project explorer window: A *project explorer window* gives you a tree-structured view of all the files inserted into the application.

TextBox Control: This control looks like a box and accepts input from the user.

Tool Box: It contains the control you placed on the form window.

User interface: A *user interface* provides a mechanism for end users to interact with the application.



Lab Exercise

1. How do I create multi-column combo box?
2. Create a label named lblAKA and containing the centered italicized word "ALIAS".
3. Create a label containing VISUAL on the first line and BASIC on the second line. Each word should be right justified.

3.9 Self Assessment Questions

1. If a TextBox's Enabled property is False, then
 - (a) The text in the box will be grayed, and the user won't be able to set focus to the TextBox.
 - (b) The text in the box will be grayed, the user can still set focus to the TextBox, but the User won't be able to make changes to the text.
 - (c) The text in the box will be grayed, and the user can still make changes to the text.
 - (d) The text in the box will appear normal, the user can set focus to the TextBox, but the user can't make any changes.

2. An access key for a TextBox control
 - (a) Can be provided in the TextBox's Caption property
 - (b) Can be provided in the TextBox's Text property
 - (c) Can be provided in an accompanying Label control
 - (d) Can be provided in the TextBox's Label Property
3. Which of these statements will assign the value of the Command Button's Caption property to the TextBox's Text property?
 - (a) Text1 = Command1
 - (b) Text1 = Command1.Caption
 - (c) Text1.Text = Command1
 - (d) Text1.Text = CStr(Command1)
4. You've been asked to author an ActiveX control for use by your company. Your ActiveX control contains a constituent label control that exposes its caption property. How could you enable the control's caption property to reflect its name when it is placed on a form at design time?
 - (a)

```
Private Sub UserControl_InitProperties()  
    LblCaption.Caption = Me.Name  
End Sub
```
 - (b)

```
Private Sub UserControl_InitProperties()  
    LblCaption.Caption = UserControl.Name  
End Sub
```
 - (c)

```
Private Sub UserControl_InitProperties()  
    LblCaption.Caption = Extender.Name  
End Sub
```
 - (d)

```
Private Sub UserControl_Initialize ()  
    LblCaption.Caption = Me.Name  
End Sub
```
5. Which two properties of a text box control are used to bind to data at runtime?
 - (a) DataField and DataMember.
 - (b) DataSource and DataMember.
 - (c) DataSource and DataField.
 - (d) You can only bind to data at design time.

6. Which cursor type should you use to populate the combo boxes?
- (a) ForwardOnly (b) Dynamic
(c) KeySet (d) Static
7. You create a new form and immediately place on it a text box, naming it txtLastName. Next, you place another text box, naming it txtFirstName. After running the application a few times, you decide you would like to change the tab order so that the cursor initially appears in txtFirstName. What is the best way to do this?
- (a) Add txtFirstName.SetFocus to the
 Form_Load() event procedure.
(b) Set txtFirstName.TabIndex = 0.
(c) Set txtFirstName.TabIndex = 1.
(d) Set txtLastName.TabStop = False.
8. In designing your company's order entry system, you've decided to use the text box control for most of the data input. One of the validation rules is to require all purchasing codes to be entered in uppercase. Which two attributes of the text box control would you use to enforce this rule?
- (a) KeyPress property (b) KeyDown event
(c) KeyPress event (d) UpperCase function
(e) UCase function (f) KeyAscii argument
9. The data entry form that you are designing will have some very strict field-level validation to ensure that accounting and other business codes are correct as you tab from field to field. Which text box event should you use to enforce this validation?
- (a) KeyPress event (b) Change event
(c) LostFocus event (d) Validate event
10. What does the following event procedure accomplish?
- ```
Private Sub cmdList_Click()
 Dim frmTest As Form
 Dim ctrTest As Control
 For Each frmTest In Forms
 For Each ctrTest In frmTest.Controls
 MsgBox frmTest.Name & " : " & ctrTest.Name
 Next
 Next
End Sub
```

- (a) All controls on all forms in the project are enumerated and displayed.
- (b) All controls on loaded forms in the project are enumerated and displayed.
- (c) Only visible controls on all forms in the project are enumerated and displayed.
- (d) Only visible controls on loaded forms in the project are enumerated and displayed.

### **3.10 Review Questions**

1. What is the difference between the Text and the Name properties of a button?
2. Place a text box on a form and select the text box. What is the effect of pressing the various arrow keys while holding down the Shift key?
3. Which of the controls presented in this section can receive the focus? Design a form containing all of the controls, and repeatedly press the Tab key to confirm your answer.
4. Create a form with two group boxes, each having two radio buttons attached to it. Run the program, and confirm that the two pairs of radio buttons operate independently of each other.
5. What is the difference between a set of check boxes attached to a group box and a set of radio buttons attached to a group box?
6. Why are radio buttons also called “option buttons”?
7. Create a text box named txtGreeting and containing the word “HELLO” in large italic letters.
8. Create a read-only text box containing the words “Visual Basic” in bold white letters on a red background.
9. Create a form named frmHello whose title bar reads “Hello World”.
10. Create a label containing a picture of a diskette. (Hint: Use the Wingdings character) Make the diskette as large as possible.
11. Create a list box that will be invisible when the program is run.
12. Create a button containing a picture of a red bell. (Hint: Use the Wingdings character %.) Make the bell as large as possible.
13. Give a situation where the MaxLength property of a text box is useful.
14. How can I call a Command button without clicking it?
15. How do I set a shortcut key for label?
16. What is the maximum size of a textbox?
17. Write a program with a single text box and a menu with the single top-level item Edit and the four second-level items Copy, Paste, Cut, and Exit. Copy should place a copy of the selected portion of txtBox into the clipboard, Paste should duplicate the contents of the clipboard at the cursor position, Cut should delete a selected portion of the text box and place it in the clipboard, and Exit should terminate the program.

### **Answers for Self Assessment Questions**

- |        |         |        |            |
|--------|---------|--------|------------|
| 1. (a) | 2. (c)  | 3. (b) | 4. (c)     |
| 5. (c) | 6. (a)  | 7. (b) | 8. (c & f) |
| 9. (d) | 10. (b) |        |            |

### **3.11 Further Reading**



*Books*

Null Dale, Michael Mc Millan, "Visual Basic".



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 4: VB Programming Fundamentals**

### **CONTENTS**

Objectives

Introduction

4.1 Data Types

4.1.1 Visual Basic Data Types

4.2 Variables

4.3 Variables Declaration

4.3.1 The Dim Statement

4.3.2 Selecting Variable Types

4.3.3 Converting Between Data Types

4.3.4 Setting Variable Scope

4.3.5 Implicit and Explicit Variable Declaration

4.3.6 Using Option Explicit Statement

4.3.7 Comparing Implicit and Explicit Variable Performance

4.3.8 Providing Names for your Variables

4.3.9 Variables Scope and Lifetime

4.3.10 Local Variables

4.3.11 Static Variables

4.3.12 Module Level Variables

4.3.13 Public vs Local Variables

4.4 Summary

4.5 Keywords

4.6 Self Assessment Questions

4.7 Review Questions

4.8 Further Readings

## **Objectives**

*After studying this unit, you will be able to:*

- Understand the various data types and its variables
- Discuss the variables and its various declaration
- Explain the Dim statement

## **Introduction**

Calculations and data storage is basic building block of any programming language. So it is very important to understand them in Visual Basic. Although the calculations themselves are quite simple, but there are some important issues to be understood along with them like how to store data, what data types are allowed in Visual Basic, where and when to declare variables and constants and most importantly the type compatibility among various data types.

### **4.1 Data Types**

Data type determines type of data variable can store. In all programming languages, you define a real-world data that refers to some specific type and needs to be understood by the compiler. The compiler of the language defines the rules for different types of data that is to be allowed. This type could be numeric, a string user defined or date. All compilers are CLS(Common Language Specification) compliant *i.e.*, it allows for certain data types. It enables cross-language programming. This interoperability ensures that if you write an application in suppose Visual Basic and the datatypes used are CLS compliant then any other language Framework can exchange information with the application. A list of Visual Basic's simple data types are given below:

#### **1. Numeric**

|                 |                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------|
| <b>Byte</b>     | Store integer values in the range of 0 - 255                                                        |
| <b>Integer</b>  | Store integer values in the range of (-32,768) - (+ 32,767)                                         |
| <b>Long</b>     | Store integer values in the range of (- 2,147,483,468) - (+ 2,147,483,468)                          |
| <b>Single</b>   | Store floating point value in the range of (-3.4x10 <sup>-38</sup> ) - (+ 3.4x10 <sup>38</sup> )    |
| <b>Double</b>   | Store large floating value which exceeding the single data type value                               |
| <b>Currency</b> | Store monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left |

#### **2. String**

Use to store alphanumeric values. A variable length string can store approximately 4 billion characters.

#### **3. Date**

Use to store date and time values. A variable declared as date type can store both date and time values and it can store date values 01/01/0100 up to 12/31/9999.



#### 4. Boolean

Boolean data types hold either a true or false value. These are not stored as numeric values and cannot be used as such. Values are internally stored as -1 (True) and 0 (False) and any non-zero value is considered as true.

#### 5. Variant

Stores any type of data and is the default Visual Basic data type. In Visual Basic if we declare a variable without any data type by default the data type is assigned as default.

##### 4.1.1 Visual Basic Data Types

| Data Type         | Suffix |
|-------------------|--------|
| Boolean           | None   |
| Integer           | %      |
| Long (Integer)    | &      |
| Single (Floating) | !      |
| Double (Floating) | #      |
| Currency          | @      |
| Date              | None   |
| Object            | None   |
| String            | \$     |
| Variant           | None   |

##### 4.1.2 Verifying Data Types

You can change a variables type with **ReDim** in Visual Basic, assign objects to variables using **Set**, and even convert standard variables into arrays. For these and other reasons, Visual Basic has a number of data verification functions, which appear in next page and you can use these functions to interrogate objects and determine their types.

Verification

function. **Dose This**

Function

|              |                                                                                |
|--------------|--------------------------------------------------------------------------------|
| Is Array ()  | Returns True if passed an array                                                |
| Is Date()    | Returns True if passed a data                                                  |
| Is Empry()   | Returns True if passed variable is uninitialized                               |
| Is Error()   | Returns True if passed an error value                                          |
| Is Missing() | Returns True if value was not passed for specified parameter in procedure call |
| Is Null()    | Returns True if passed NULL                                                    |
| Is Numeric() | Returns True if passed a numeric value                                         |
| Is Object()  | Returns True if passed an object                                               |



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

## 4.2 Variables

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 has ended. If you want to save the values of variables or other data. Before allocating storage space for a variable, decide what the variable's lifetime will be, or in other words, which procedures and which modules should have access to the variable's value.

- Procedure-level variables are created with a Dim statement placed right in the procedure where it's going to be used. The value of a procedure level variable cannot be accessed outside it's procedure. When the procedure finishes (End Sub or End Function), the variable is destroyed and memory allocated to the variable is released.
- Module-level variables are created with a Private statement in the general declarations section of a Form or code module. The value of the module level variable is available to every procedure in that module. Memory allocated to the module-level variable is not destroyed until the module is Unloaded.
- Global variables are created with a Public statement in the general declarations section of a Form or code module. The value of a Global variable is available to any procedure, in any Form or code module. Memory allocated to a Global variable is not released until your program shuts down.

It would certainly be easier to make every variable Global. You wouldn't have to think twice about it's availability at any given time. But sometimes, every byte of memory counts, so don't give your variables any more life than they actually require.



A variable name must begin with an alphabet letter and should not exceed 255 characters. It must be unique within the same scope. It should not contain any special character like %, &, !, #, @ or \$.

## 4.3 Variables Declaration

Before using variables, you have to set aside memory space for them\_after all, that\_s what they are, locations in memory. Usually, you use the **Dim** statement to declare variables, although you can also use the **Private** (declare a private variable), **Public** (declare a global variable), **Static** (declare a variable that holds its value between procedure calls), **ReDim** (redimension a dynamic array), or **Type** (declare a user-defined type) keywords to declare variables, as we'll see in the tasks covered in this chapter.

### 4.3.1 The Dim Statement

Here is how you use the **Dim** statement:

```
Dim [WithEvents] varname([subscripts]) [As [New] type] [, [WithEvents]
```

*varname* [(*subscripts*)] [As [New] *type*] . . .

The **WithEvents** keyword is valid only in class modules. This keyword specifies that *varname* is an object.

Variable used to respond to events triggered by an ActiveX object. The *varname* identifier is the name of the variable you are declaring. You use *subscripts* if you are declaring an array.

You set up the *subscripts* argument this way:

[*lower* To] *upper* [, [*lower* To] *upper*]



In Visual Basic, you may declare up to 60 dimensions for an array.

The **New** keyword enables creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it. This means you don't have to use the **Set** statement to assign the object reference. Here is an example:

```
Dim DataSheet As New Worksheet
```

The *type* argument specifies the data type of the variable, which may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (for variable-length strings), **String \* length** (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. If you don't specify a type, the default is **Variant**, which means the variable can act as any type.



By default in Visual Basic, numeric variables are initialized to 0, variable-length strings are initialized to a zero-length string (`__`), and fixed-length strings are filled with zeros. Variant variables are initialized to **Empty**.

Here is an example of declaring variables using **Dim**:

```
Dim EmployeeID As Integer
```

```
Dim Employee Name As String
```

```
Dim Employee Address As String
```



*Did u know?*

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

### 4.3.2 Selecting Variable Types

It is time to create a new variable but what type should you use? For that matter, exactly what type of variable types are there and what do they do? Even if you remember what types there are, you probably won't remember the range of possible values that variable type allows.

There is a wide range of data types, so we'll use a table here. The Visual Basic variable types appear in Table 4.1 for reference, making selecting the right type a little easier (note that although Visual Basic lists a **Decimal** variable type, that type is not yet actually supported). We also include the literal suffix symbols for numeric values in Table 4.1 those are the suffixes you can add to the end of values or variables to tell Visual Basic their type, like `strUserFormatString$`.

Table 4.1

| Variable types            |          |         |                                                                                                                                                             |
|---------------------------|----------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Variable                  | Bytes of | Literal |                                                                                                                                                             |
| Type                      | Storage  | Suffix  | Range                                                                                                                                                       |
| Boolean                   | 2        | N/A     | True, False                                                                                                                                                 |
| Byte                      | 1        | N/A     | 0 to 255                                                                                                                                                    |
| Currency                  | 8        | @       | – 922,337.203,685,477,5808 to<br>222,337,203,685,477,5807                                                                                                   |
| Date                      | 8        | #&#     | 1 January 100 to 31 December 9999 and times<br>from 0:00:00 to 23:59:59                                                                                     |
| Decimal                   | 12       | N/A     | – 79,228,162,514,264,337,593,543,950,335 to<br>79,228,162,514,264,337,593,543,950,335                                                                       |
| Double                    | 8        | #       | – 1.79769313486232E308 to<br>– 4.94065645841247E–324 for negative values<br>and from 4.94065645841247E–324 to<br>1.79769313486232E for positive values      |
| Integer                   | 2        | %       | – 32,768 to 32,767                                                                                                                                          |
| Long                      | 4        | &       | – 2,147,483,483,648 to 2,147, 483,647                                                                                                                       |
| Object                    | 4        | N/A     | N/A                                                                                                                                                         |
| Single                    | 4        | !       | – 3,403823E to –1,401298–45 for negative<br>values and from 1.4012298E –45 to 3.402823E38<br>for positive values                                            |
| String                    | N/A      | \$      | A variable-length string can contain up to<br>approximately 2 billion characters; a fixed-length<br>string can contain 1 to approximately 64K<br>characters |
| User-defined<br>data type | N/A      | N/A     | N/A                                                                                                                                                         |
| Variant                   | N/A      | N/A     | N/A                                                                                                                                                         |

As you can see in Table, Visual Basic has a large number of data formats. The **Variant** type deserves special mention, because it is the default variable type. If you don't declare a type for a variable, it is made a variant:

```
Private Sub Command1_Click()
Dim NumberTrains
...
End Sub
```

In this case, the variable **NumberTrains** is a variant, which means it can take any type of data. For example, here we place an integer value into **NumberTrains** (note that we specify that 5 is an integer by using the percent sign [%] suffix as specified in Table 3.1):

```
Private Sub Command1_Click()
Dim NumberTrains
NumberTrains = 5%
End Sub
```

We could have used other data types as well; here, for example, we place a string into **NumberTrains**:

```
Private Sub Command1_Click()
Dim NumberTrains
NumberTrains = "Five"
End Sub
```

And here we use a floating point value (! is the suffix for single values):

```
Private Sub Command1_Click()
Dim NumberTrains
NumberTrains = 5.00!
End Sub
```

Be careful of variants, however; they waste time because Visual Basic has to translate them into other data types before using them, and they also take up more space than other data types.

### 4.3.3 Converting Between Data Types

Visual Basic supports a number of ways of converting from one type of variable to another in fact, that is one of the strengths of the language.

Visual Basic

data

conversion Use This

function

To Do This

ANS value

to string

String to

lowercase or

uppercase

Chr

Data to

serial

Format. L Case. U Case

number

Decimal

Data Serial. Data Value

|              |                                                          |
|--------------|----------------------------------------------------------|
| number to    | Chr                                                      |
| other bases  |                                                          |
| Number to    | Format. L Case. U Case                                   |
| string       |                                                          |
| One data     | Data Serial. Data Value                                  |
| type to      |                                                          |
| another      | Hex. Oct                                                 |
| Data to day. |                                                          |
| month,       | Format, Str                                              |
| weekday, or  |                                                          |
| year         | CBool, CByte, CCur, CDate, CDbt, CDec, CInt, Clug, CSug, |
| Time to      | C Str, CVar, CVer, Fix Int                               |
| hour         |                                                          |
| minute, or   | Day, Month, Week, year                                   |
| second       |                                                          |
| string to    |                                                          |
| ASCII value  | Hour, Minute, Second                                     |
| String to    | Ass                                                      |
| number       | Val                                                      |
| Time to      |                                                          |
| serial       | Time Serial, Time Value                                  |
| number       |                                                          |



Did u know?

Note that you can cast variables from one type to another in Visual Basic using the functions **CBool()**, **CByte()**, and so on.

#### 4.3.4 Setting Variable Scope

You have just finished creating a new dialog box in your greeting card program, and it is a beauty. However, you realize there is a problem: the user enters the new number of balloons to display the greeting card in **TextBox1** of the dialog box, but how do you read that value in the rest of the program when the user closes the dialog box?

It is tempting to set up a global variable, **intNumberBalloons**, which you fill in the dialog box when the user clicks on the OK button. That way, you will be able to use that variable in the rest of the program when the dialog box is closed. But in this case, you should resist the

temptation to create a global variable it is much better to refer to the text in the text box this way (assuming the name of the dialog form you have created is **Dialog**):

```
intNumberBalloons = Dialog.TextBox1.Text
```

This avoids setting up a global variable needlessly. In fact, one of the most important aspects of Visual Basic programming is *variable scope*. In general, you should restrict variables to the smallest scope possible.

There are three levels of variable scope in Visual Basic, as follows:

“ Variables declared in procedures are private to the procedure.

“ Variables declared at the form or module level in the form or modules (General) section using **Dim**, **ReDim**, **Private**, **Static**, or **Type** are form- or module-level variables. These variables are available throughout the module.

“ Variables declared at the module level in the modules (General) section using **Public** are global and are available throughout the project, in all forms and modules. Note that you cannot use **Public** in procedures.



If you use the **Option Private Module** statement in a module or form, all variables in the module or form become private to the module, no matter how they are declared.

### 4.3.5 Implicit and Explicit Variable Declaration

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

Syntax

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

For example,

```
Dim strName As String
```

```
Dim intCounter As Integer
```

### 4.3.6 Using Option Explicit Statement

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

```
Intcount = Intcount + 1
```

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

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

#### Option Explicit

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

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

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

### **4.3.7 Comparing Implicit and Explicit Variable Performance**

The default data type for Visual Basic variables is the variant. This means that, unless you specify otherwise, every variable in your application will be a variant. The data type is not very efficient. Its data storage requirements are greater than the equivalent simple data type. The computer spends more time keeping track of the data type contained in a variant than for other data types.

### **4.3.8 Providing Names for Your Variables**

Visual Basic provides extremely liberal rules for naming variables. Briefly, all variable names must conform to the following requirements:

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

### **4.3.9 Variables Scope and Lifetime**

A variable is more than just a simple data repository. Every variable is a dynamic part of the application and can be used at different times during the program's execution. The declaration of a variable establishes more than just the name and data type of the variable. Depending on the keyword used to declare the variable and the placement of the variable's declaration in the program's code, the variable might be visible to large portions of the application's code. Alternatively, a different placement might severely limit where the variable can be referenced in the procedures within the application.

The visibility of a variable or procedure is called its scope. A variable that can be seen and used by any procedure in the application is said to have public scope. Another variable, one that is usable by a single procedure, is said to have scope that is private to that procedure.

There are many analogies for public and private scope. Variables declared within a procedure are local to that procedure and cannot be used or referenced outside that procedure.

In each case, the Dim keyword was used to define the variable.



Dim is shorthand for dimension and is a rather archaic expression that instructs Visual Basic to allocate enough memory to contain the variable that follows the Dim keyword. Therefore, Dim i As Integer allocates less memory (2 bytes) than Dim s As Double (8 bytes). There is no way to make a variable declared within a procedure visible outside of that procedure.

The public keyword is used to make a variable visible throughout an application. Public can be used only at the module level and cannot be used within a procedure. Usually, the Public keyword is used only in standard modules that are not part of a form. Every variable declared in the general section of the standard module is public throughout the application unless the Private keyword is used. Private restricts the visibility of a variable to the module in which the variable is declared.

#### 4.3.10 Local Variables

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

```
Dim sum As Integer
```

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

#### 4.3.11 Static Variables

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

```
Static intPermanent As Integer
```

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

```
Function RunningTotal ()
```

```
Static Accumulate
```

```
Accumulate = Accumulate + num
```

```
RunningTotal = Accumulate
```

```
End Function
```

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

```
Static Function RunningTotal ()
```



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

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

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



Make a visual basic program use static Variable.

#### 4.3.12 Module Level Variables

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

In the declarative section of any module

```
Private LoginTime As Date ' A private module-level variable
```

```
Dim LoginPassword As String ' Another private module-level variable
```

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

In the declarative section of Form1 module

```
Public CustomerName As String ' A Public property
```

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

From outside Form1 module...

```
Form1.CustomerName = "Pradip"
```

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

is re-created, Visual Basic reallocates memory for all variables and resets them to their default values (0 for numeric values, "" for strings, Nothing for object variables).

#### 4.3.13 Public vs Local Variables

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



##### *Case Study*

#### **Success Story of Programming**

##### *Overview of Programming Language*

The history of programming language gives us the complete details of gradual development of programming language since its origin. The idea of computer programming language was based on the concept of difference engine of Charles Babbage. It was in the year 1822 that this difference engine was used for the first time to transfer task related command to the computer system. History of programming language shows that programming language has undergone substantial changes over the last few decades.

##### *Earliest Forms of Programming*

Initially the difference engine of Charles Babbage was used as gears to perform the quantitative tasks. Physical motion was the primitive version of computer language. Following the introduction of ENIAC in the year 1942, the electric signals started its journey in the field of computer programming. The electric signals followed the same mechanism used by difference engine. However, the difference between the two lies in their modes of operation.

##### *Modern Versions of Programming Language*

The history of programming reveals that the modern version of programming language was launched in 1957 in FORTRAN format. FORTRAN, designed by IBM, served as a medium for translating formula. However this form of programming language was the simplest of all the versions used so far.

The Algol language introduced in the following year was a major breakthrough in the history of programming. Algol has contributed substantially in the development of modern programming languages like, C, C++ and Pascal. The programming language Pascal, presents a unique combination of FORTRAN, ALGOL, COBOL and some other commonly used languages.

#### **Questions**

1. Explain the concept of language.
2. Give some Modern versions of Programming language.

#### 4.4 Summary

- Variables are temporary storage space for numbers, text and objects.
- The compiler of language defines the rule for different types of data.
- Visual basic has a number of data verification function.
- Temporary storage of numbers, text and objects are the variables.
- Static variables are also ideal for making controls alternately visible/invisible.

#### 4.5 Keywords

**Common Language Specification (CLS):** All compilers are CLS(Common Language Specification) compliant.

**Dim statement:** It declares variable. It uses with events keywords.

**Global variables:** Global variables are created with a Public statement in the general declarations section of a Form or code module.

**Local variables:** It is declared inside a procedure. It exists as long as the procedure in which they are declared, is executing.

**Module-level variables:** Module-level variables are created with a Private statement in the general declarations section of a Form or code module.

**Procedure-level variables:** Procedure-level variables are created with a Dim statement placed right in the procedure where it's going to be used.

**Static Variables:** They are not reinitialized each time Visual Invokes a procedure and therefore retains or preserves value even when a procedure ends.



Lab Exercise

1. How can I find the strings after compilation?
2. How do I search for files and replace strings within them?

#### 4.6 Self Assessment Questions

1. A global variable is visible to a watch of which type of scope?  
(a) Global (b) Module-level  
(c) Procedure-level (d) Unbound
2. A module-level variable is visible to a watch of which type of scope?  
(a) Global (b) Module-level  
(c) Procedure-level (d) Unbound
3. The highest Watch scope meaningful to a variable declared in a form using the Public keyword is  
(a) Global (b) Module-level  
(c) Procedure-level (d) Unbound

4. The highest Watch scope meaningful to a variable declared in a module using the Private keyword is
  - (a) Global
  - (b) Module-level
  - (c) Procedure-level
  - (d) Indeterminate
5. The with Events Keyword is valid only in class modules.
  - (a) False
  - (b) True

#### **4.7 Review Questions**

1. Dim x, y as integer. What is x and y data type?
2. What are the scopes of the variables?
3. Explain about declaring variables in Visual basic.
4. Describe the different scopes of variables in VB.
5. What is Implicit?
6. What is Explicit?
7. What is variable scope lifetime?

#### **Answers for Self Assessment Questions**

1. (a) (b) and (c)
2. (b) and (c)
3. (a)
4. (b)
5. (b)

#### **4.8 Further Readings**



Books

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Milan, Visual Basic .Net.



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 5: VB String and Operators**

### **CONTENTS**

Objectives

Introduction

5.1 Strings

5.1.1 Variables and Strings

5.1.2 Fixed-Length Strings

5.1.3 Working with Fixed Length Strings

5.1.4 Testing for Fixed Length Strings

5.1.5 Size String

5.1.6 Path Compact Path Ex

5.1.7 Trim to Null

5.1.8 Trim to Char

5.2 Operators-math Operators

5.2.1 Operators-Math Operator

5.2.2 Multiplication Operator

5.2.3 Division Operator

5.2.4 Exponents

5.2.5 Handling Operators And Operator Precedence

5.2.6 Using IF & Else Statements

5.2.7 Using Select Case

5.2.8 Making Selections with Switch() And Choose()

5.2.9 The Choose() Function

5.2.10 Sending Keystrokes to other Programs

5.2.11 Handling Higher Math

5.2.12 Handling Dates And Times

5.3 Concatenation Operators

5.3.1 Differences Between the Two Concatenation Operators

5.4 Logical Operator

5.5 Summary

5.6 Keywords

5.7 Self Assessment Questions

5.8 Review Questions

5.9 Further Reading

## **Objectives**

*After studying this unit, you will be able to:*

- Understand string in visual basic
- Discuss operator and math operators
- Explain concatenation operators
- Understand logical operator

## **Introduction**

A string is one or a combination of characters. To declare a variable for it, you can use either the string or the Object data types. To initialize the variable, put its value in double-quotes and assign it to the variable. Here are examples:

```
Public Module Exercise

 Public Function Main() As Integer

 Dim First Name As Object

 Dim Last Name As String

 First Name = "William"

 Last Name = "Sansen"

 Return 0

 End Function

End Module
```

An Operator is a code element that performs an operation on one or more code elements that hold values. Value elements include variables, constants, literals, properties, returns from Function and Operator procedures, and expressions.

## **5.1 Strings**

The most common types of data processed by Visual Basic are numbers and strings. Sentences, phrases, words, letters of the alphabet, names, telephone numbers, addresses, and Social Security numbers are all examples of strings. Formally, a String literal is a sequence of characters that is treated as a single item. String literals can be assigned to variables, displayed in text boxes and list boxes, and combined by an operation called concatenation (denoted by &).

### **5.1.1 Variables and Strings**

A String variable is a name used to refer to a String. The allowable names of String variables are the same as those of numeric variables. The value of a String variable is assigned or altered with assignment statements and displayed in a list box like the value of a numeric variable. String variables are declared with statements of the form

Dim varName As String



Example 1: The following program computes the sum of two numbers supplied by the user:

| Object       | Property       | Setting     |
|--------------|----------------|-------------|
| frmAdd       | Text           | Addition    |
| lblFirstNum  | AutoSize       | False       |
| Text         | First Number:  |             |
| txFirstNum   |                |             |
| lblSecondNum | AutoSize       | False       |
| Text         | Second Number: |             |
| btnCompute   | Text           | Compute Sum |
| txtSecondNum |                |             |
| lblSum       | Text           | Sum:        |
| txtSum       | ReadOnly       | True        |

*Private Sub btnCompute\_Click() Handles btnCompute.Click*

*Dim num1, num2, sum As Double*

*num1 = CDbI(txtFirstNum.Text)*

*num2 = CDbI(txtSecondNum.Text)*

*sum = num1 + num2*

*txtSum.Text = CStr(sum)*

*End Sub*

[Run, type 45 into the first text box, type 55 into the second text box, and click on the button.]



### 5.1.2 Fixed-Length Strings

Fixed length strings are automatically filled with spaces to pad them to their fixed-length. When working with fixed-length strings, you can use RTrim\$ to trim the extra spaces.

If a fixed-length string is declared, it will be filled with Null characters until it is used. The Visual Basic trim functions (RTrim\$, LTrim\$, and Mid\$) will not trim Null characters, so be sure to assign a fixed-length string to an empty string immediately.

```
Din Mystring As String* 10
```

```
My String = "Chilkat" 'Prints "Chikat"
```

```
Print RTrim$ (My String) 'Prints "Chilkat"
```

```
Din Another String As String * 10
```

```
Print Len (RTrim$ (Another String)) 'Prints 10, because the string Contains Null characters.
```

### 5.1.3 Working With Fixed Length Strings

Strings in VBA are by their nature resizable. All of VBA's string functions are used to set or retrieve portions of strings of variable length. You may find, however, that you need strings of a specific length, such as when you are writing text-based files that will be parsed by another application, or simply for alignment in list boxes.

You can declare a string to be a fixed length with a declaration like

```
Dim S As String * 20
```

This declares a string of 20 characters. The disadvantage of this method is that the length of the string is always fixed at 20. You cannot resize the string "on the fly" at run time. You can neither lengthen nor shorten the string. Moreover, you must know the length of the string at design-time. You can't create fixed length strings at run time. The SizeString function on this page can be used to create pseudo-fixed length strings. The strings are normal, sizable strings, by are sized a specified length, containing the specified text value on either the left or right region of the string.

The following are a few VBA procedures for working with strings. These functions will work in any version of Office and any application. They will also work in Visual Basic 6. There is nothing in the code that is specific to Excel.

You can download a bas module containing these functions [here](#).

### 5.1.4 Testing For Fixed Length Strings

At times, you may need to determine whether a string variable is a fixed length string or a sizable string. You cannot do this with a function procedure because VBA will convert fixed length strings to sizable strings when passing the string variable to a procedure. Instead, you must write code within the procedure in which the string is declared (unless it is a module-level or global variable) to test whether a string is fixed or sizable. The following code will do this.

```
Dim S As String ' normal, sizable string
```

```
Dim FS As String * 10 ' fixed length string, 10 chars
```

```
Dim OrigLen As Long ' original length of string
```

```
Dim OrigVal As String ' original value of string
```

```
////////////////////////////////////
```

```
' put some arbitrary values in FS And S.
' This is for demonstration purposes only.
' It is not required for the code to work.
////////////////////////////////////

FS = "ABC" ' length of FS is still 10 chars, even though it contains only 3 chars.
S = "DEF" ' length of S is 3 chars.
////////////////////////////////////

' test whether FS is fixed length string
////////////////////////////////////

OrigLen = Len(FS) ' length of FS string
OrigVal = FS ' save original value of FS
FS = FS & " " ' attempt to append a space to the end of FS
If OrigLen = Len(FS) Then ' if the length of FS didn't change, the string is fixed length
 Debug.Print "FS is a fixed length string"
Else ' if the length of FS did change, the string is sizable
 Debug.Print "FS is a sizable string"
'restore original value
 FS = OrigVal
End If
////////////////////////////////////

' test whether S is fixed length string
////////////////////////////////////

OrigLen = Len(S) ' save length of S
OrigVal = S ' save original value
S = S & " " ' attempt to append a space to the end of S
If OrigLen = Len(S) Then ' if length of S didn't change, the string if fixed length
 Debug.Print "S is a fixed length string"
Else' if the length of S did change, the string is sizable
 Debug.Print "S is a sizable string"
'restore original value
 S = OrigVal
End If

The output in the Immediate Window in VBA of the code above is:
FS is a fixed length string
S is a sizable string
```

### **5.1.5 SizeString**

The following VBA procedure will return a string variable containing the specified text, on either the left of the right, padded with PadChar on either the right or left to make a string Length characters long.

```

' =====
' This enum is used by SizeString
' to indicate whether the supplied text
' appears on the left or right side of
' result string.
' =====

Public Enum SizeStringSide
 TextLeft = 1
 TextRight = 2
End Enum

Public Function SizeString(Text As String, Length As Long, _
 Optional ByVal TextSide As SizeStringSide = TextLeft, _
 Optional PadChar As String = " ") As String
 ' =====
 ' SizeString
 ' This procedure creates a string of a specified length. Text is the original string
 ' to include, and Length is the length of the result string. TextSide indicates whether
 ' Text should appear on the left (in which case the result is padded on the right with
 ' PadChar) or on the right (in which case the string is padded on the left). When padding on
 ' either the left or right, padding is done using the PadChar. character. If PadChar is omitted,
 ' a space is used. If PadChar is longer than one character, the left-most character of PadChar
 ' is used. If PadChar is an empty string, a space is used. If TextSide is neither
 ' TextLeft or TextRight, the procedure uses TextLeft.
 ' =====

 Dim sPadChar As String
 If Len(Text) >= Length Then
 'if the source string is longer than the specified length, return the
 ' Length left characters
 SizeString = Left(Text, Length)
 Exit Function
 End If
 If Len(PadChar) = 0 Then
 ' PadChar is an empty string. use a space.
 sPadChar = " "
 Else
 ' use only the first character of PadChar
 sPadChar = Left(PadChar, 1)
 End If
 If (TextSide <> TextLeft) And (TextSide <> TextRight) Then

```

```
'if TextSide was neither TextLeft nor TextRight, use TextLeft.
TextSide = TextLeft
End If
If TextSide = TextLeft Then
'if the text goes on the left, fill out the right with spaces
SizeString = Text & String(Length - Len(Text), sPadChar)
Else
'otherwise fill on the left and put the Text on the right
SizeString = String(Length - Len(Text), sPadChar) & Text
End If
End Function
```

### 5.1.6 PathCompactPathEx

The converse of the SizeString is the PathCompactPathEx Windows API function. While SizeString is designed to increase the length of a string by padding with a character, the PathCompactPathEx is used shorten a string to a specified length. PathCompactPathEx is intended to trim fully qualified filenames to a specified number of characters, omitting or truncating a folder name component of the full file name, but it can be used with any text string. When used with a fully qualified file name, it omits part of the file name (such as a folder name) to size the string to fit in the specified number of characters, replacing the deleted text with the characters "...". For example PathCompactPathEx with a length parameter of 30 will shorten the Excel Application's file name to a truncated string. It will change

C:\Program Files\Office 2003\OFFICE11\Excel.exe

to

C:\Program Files...\Excel.exe

When PathCompactPathEx is used with arbitrary text without '\' characters, it typically just truncates the InputString, removing the characters on the right and replacing them with "...". The following function, ShortenTextToChars, is a wrapper function for PathCompactPathEx that handles the variable and string buffer handling, and most importantly the validation of the NumberOfCharacters parameter. If you pass an invalid NumberOfCharacters value to PathCompactPathEx, it will cause the application to crash. PathCompactPathEx is unforgiving of any invalid input parameters. The ShortenTextToChars function uses the GetSystemErrorMessageText function, available here and the TrimToNull function available here.

```
Private Declare Function PathCompactPathEx Lib "shlwapi.dll" _
Alias "PathCompactPathExA" (_
ByVal pszOut As String, _
ByVal pszSrc As String, _
ByVal cchMax As Long, _
ByVal dwFlags As Long) As Long
Public Function ShortenTextToChars(InputText As String, _
NumberOfCharacters As Long) As String
```

////////////////////////////////////

```
' ShortenTextToChars
' This function returns a shortened version of the InputText parameter that is
' NumberOfCharacters in length. This function is primarily designed for use with
' fully-qualified file names. With a file name, the function will remove or truncate
' an element of the path (e.g., a folder name), replacing the removed text with the string
' "...". While this is intended for use with file names, it will work with any text string.
' When used on text that does not contain '\' characters, it typically just truncates the
' right side of InputText.
' Returns vbNullString is an error occurred.
'///

Dim ResString As String
Dim Res As Long
Dim ErrorNumber As Long
Dim ErrorText As String
'///

' Ensure that InputText is not an empty string
'///

If InputText = vbNullString Then
MsgBox "The InputText parameter is an empty string"
ShortenTextToChars = vbNullString
Exit Function
End If
'///

' Test for NumberOfCharacters <= 3. If the InputText
' is 3 or fewer characters, PathCompactPathEx would replace the
' entire string with "...". We don't want that. Return the entire
' InputText.
'///

If Len(InputText) <= 3 Then
ShortenTextToChars = InputText
Exit0 Function
End If

' Test for NumberOfCharacters less than equal to 3.
' If the NumberOfCharacters <= 3, PathCompactPathEx would replace
' the entire InputString with "...".
' Instead, return the left-most characters and get out.
'///

If NumberOfCharacters <= 3 Then
```

ShortenTextToChars = Left(InputText, NumberOfCharacters)

Exit Function

End If

////////////////////////////////////

' Ensure we have a valid number of characters. If NumberOfCharacters  
' is less than or equal to 0, or greater than the length of  
' the InputText, PathCompactPathEx will crash the application.

////////////////////////////////////

If NumberOfCharacters <= 0 Then

MsgBox "The NumberOfCharacters must be greater than 0."

ShortenTextToChars = vbNullString

Exit Function

End If

////////////////////////////////////

' Here we test if the length of InputTex is equal to  
' NumberOfCharacters. If they are equal, return the  
' the entire string and exit. If we allowed  
' PathCompactPathEx to process the string, it would truncate  
' on the right and replace the last three characters on the  
' right with "...". We don't want that behavior — we  
' want to return the entire string.

////////////////////////////////////

If Len(InputText) = NumberOfCharacters Then

ShortenTextToChars = InputText

Exit Function

End If

////////////////////////////////////

' Initialize the buffer. When PathCompactPathEx  
' creates its string, it considers NumberOfCharacters  
' to include room for the trailing null character. Thus  
' the actual number of real characters it returns will be  
' NumberOfCharacters-1. Thus, we allocate the string  
' to NumberOfCharacters+2 = 1 because we want  
' NumberOfCharacters (without the trailing null)  
' returned, + 1 for trailing null.

////////////////////////////////////

```
ResString = String$(NumberOfCharacters + 2, vbNullChar)
' Shorten the text with PathCompactPathEx

Res = PathCompactPathEx(ResString, InputText, NumberOfCharacters, 0&)
If Res = 0 Then
 ErrorNumber = Err.LastDllError
 ErrorText = GetSystemErrorMessageText(ErrorNumber)
 MsgBox "An error occurred with PathCompactPathEx" & vbCrLf & _
 "Error Number: " & CStr(ErrorNumber) & vbCrLf & _
 "Description: " & ErrorText
 ShortenTextToChars = vbNullString
Exit Function
End If

' trim to get the characters to the left
' of the vbNullChar character.

ResString = TrimToNull(Text:=ResString)

return the result string

ShortenTextToChars = ResString
End Function
```

### 5.1.7 TrimToNull

When you are calling Windows API functions, you are often required to pass in a string variable that will be populated with the result of the function. An example is the *GetWindowsDirectory* function. In these cases, you must first initialize the string to a given length (often the length is defined by the constant `MAX_PATH = 260`, a Windows-mandated value of the maximum length of a fully-qualified file name). You can use either the `String$` or `Space$` function to initialize the string to a given length:

```
Public Const MAX_PATH = 260 ' Windows mandated value

StrValue = String$(MAX_PATH,vbNullChar) ' initialize StrValue to MAX_PATH vbNullChars.
' or

StrValue = Space$(MAX_PATH) ' initialize StrValue to MAX_PATH spaces.
```

When the API call returns, the buffer remains at its original length (the API functions never resize the result buffer). The API function will put a null character (`Chr(0)` or `vbNullChar`) at the end

of the actual data in the string variable. It is up to you as the programmer to extract the data to the left of the vbNullChar. The following function will do this for you. This is a very simple function, but if you use a lot of API calls in your code, it is worthy of its own function.

```
Public Function TrimToNull(Text As String) As String
```

```
////////////////////////////////////
```

```
' TrimToNull
```

```
' This function returns the portion of Text that is to the left of the vbNullChar
```

```
' character (same as Chr(0)). Typically, this function is used with strings
```

```
' populated by Windows API procedures. It is generally not used for
```

```
' native VB Strings.
```

```
' If vbNullChar is not found, the entire Text string is returned.
```

```
////////////////////////////////////
```

```
Dim Pos As Integer
```

```
Pos = InStr(1, Text, vbNullChar)
```

```
If Pos > 0 Then
```

```
TrimToNull = Left(Text, Pos - 1)
```

```
Else
```

```
TrimToNull = Text
```

```
End If
```

```
End Function
```

With a little extra code, TrimToNull can be expanded to trim to any character or string of characters.

### **5.1.8 TrimToChar**

The following function, TrimToChar, will return the text to the left of either the first or last occurrence of a specified character or string of characters.

```
Public Function TrimToChar(Text As String, TrimChar As String, _
```

```
Optional SearchFromRight As Boolean = False) As String
```

```
////////////////////////////////////
```

```
' TrimToChar
```

```
' This function returns the portion of the string Text that is to the left of
```

```
' TrimChar. If SearchFromRight is omitted or False, the returned string
```

```
' is that string to the left of the FIRST occurrence of TrimChar. If
```



```
' SearchFromRight is True, the returned string is that string to the left of the
' LAST occurrence of TrimToChar. If TrimToChar is not found in the string,
' the entire Text string is returned. TrimChar may be more than one character.
' Comparison is done in Text mode (case does not matter).
' If TrimChar is an empty string, the entire Text string is returned.
```

////////////////////////////////////

```
Dim Pos As Integer
' Test to see if TrimChar is vbNullString. If so, return the whole string. If we
' don't test here and used InStr as in the next logic block, an empty string would
' be returned.
If TrimChar = vbNullString Then
TrimToChar = Text
Exit Function
End If
' find the position in Text of TrimChar
If SearchFromRight = True Then
' search right-to-left
Pos = InStrRev(Text, TrimChar, -1, vbTextCompare)
Else
' search left-to-right
Pos = InStr(1, Text, TrimChar, vbTextCompare)
End If
' return the sub string
If Pos > 0 Then
TrimToChar = Left(Text, Pos - 1)
Else
TrimToChar = Text
End If
End Function
```



Perhaps the biggest bottleneck is that VB makes copies of the string data when doing some of the operations. Even when you're just reading strings (and not planning to make any modifications), you can easily end up making a large number of copies. The copying costs you time if string processing is an intensive part of your program. Another reason is that some of the widely used functions are implemented in a non-straightforward way. They may be doing more work than what is required for your task. Fortunately, you can often replace an advanced functions with a simpler and faster alternative.

## **5.2 Operators-Math Operator**

### **3.5.1 Using Addition and Subtraction Operators**

The two simplest math operations are addition and subtraction. If you have ever used a calculator, you already have a good idea how these operations are performed in a line of computer code.

A computer program, however, gives you greater flexibility than a calculator in the operations you can perform. Your programs aren't limited to working with literal numbers (for example, 1, 15, 37, 63, and -105.2). Your program can add or subtract two or more literal numbers, numeric variables, or any functions that return a numeric value. Also, like a calculator, you can perform per form the operations in you program.

The operator for addition in visual Basic is the plus sign (+). The general use of this operator is as follows:

`Result=iNumber One + iNuber Two + iNuberThree`

Result is a variable (or control property) that contains the sum of the three numbers to the right of the equal sign (=). The equal sign indicates the assignment of a value to the variable. iNumberone, iNumberTwo, and iNumberThree are numeric variables. As just described, you can also add literal numbers or return values rather than numeric variables from a function. You can add as many numbers together as you like, but each number pair must be separated by a plus sign.

The operator for subtraction is the minus sing (-). The syntax is basically the same as for addition:

`Result=NumberOne-NumberTwo-NumberThree`

Although the order doesn't matter in addition, in subtraction the number to the right of the minus sign is subtracted from the number to the left of the sign. If you have multiple numbers, the second number is subtracted from the first, the third number is subtracted from that result, and so on, moving from left to right. For example, consider the following equation:

`Result=15-603`

The computer first subtracts 6 from 15 to yield 9. It then subtracts 3 from 9 to yield 6, which is the final answer stored in the variable Result. This left-to-right processing is used in the evaluation of expressions whenever similar operations are being performed.

If you really want to have 3 subtracted from 6 before it's subtracted from 15 for a Result of 12, you have to change the operator or force the program to evaluate the expression in the order you intended. (This order of execution is covered later in the section "Setting the Order of Precedence in Statements.") For now, consider the following statements and their resulting values:

1. `Result = 15-"3"-"6` 'results in 6
2. `Result = 15-"6+3` 'results in 12
3. `Result = 3-6+15` 'results in 12

After having studied addition and subtraction operators, we will take a look at the multiplication, division and exponent operators in this article.

### 5.2.2 Multiplication Operator

We simply use the multiplication operator—the asterisk (\*)—to multiply two or more numbers. The syntax of a multiplication statement is almost identical to the ones used for addition and subtraction, as follows:

```
Result = NumberOne * NumberTwo * NumberThree
```

As before, Result is the name of a variable used to contain the product of the numbers being multiplied, and NumberOne, NumberTwo, and NumberThree are numeric variables. Again, you also can use literal numbers or a return value from a function.

### 5.2.3 Division Operator

Division in Visual Basic is a little more complicated than multiplication. Last article showed three division operators. The most common one is `/`. This type is known as floating-point division (the normal type of division). This type of division returns a number with its decimal portion, if one is present.

Visual Basic supports two other ways to divide numbers: integer division and modulus (or remainder) division.

Integer division divides one number into another, and then returns only the integer portion of the result. The operator for integer division is the backward slash (`\`):

```
Result = NumberOne \ NumberTwo
```

```
Result = 2.3 \ 2 'The value of Result is 1
```

Modulus or remainder division divides one number into another and returns what's left over after you obtain the largest integer quotient possible. The modulus operator is the word `Mod`, as follows:

```
Result = NumberOne Mod NumberTwo
```

```
Result = 11 Mod 3 'result in 2
```

```
'(11/3 = 3 with a remainder of 2)
```

### 5.2.4 Exponents

Exponents also are known as powers of a number. For example, 2 raised to the fourth power is equivalent to  $2 \times 2 \times 2 \times 2$ , or 16. Exponents are used quite a lot in computer operations, in which many things are represented as powers of two. Exponents also are used extensively in scientific and engineering work, where many things are represented as powers of 10 or as natural logarithms. Simpler exponents are used in statistics, in which many calculations depend on the squares and the square roots of numbers.

To raise a number to a power, you use the exponential operator, a caret (^). Exponents greater than one indicate a number raised to a power. Fractional exponents indicate a root, and negative exponents indicate a fraction. The following is the syntax for using the exponential operator:

```
Result = NumberOne ^ Exponent
```



Example:  $2^4$  'result is 16

*In the next article, we will try to see the hierarchy of mathematical operations.*

### 5.2.5 Handling Operators And Operator Precedence

You have done well in your computer class, so well that the instructor has asked you to calculate the average grade on the final. Nothing could be easier, you think, so you put together the following program:

```
Private = Sub Command1_Click()
Dim intGrade1, intGrade2, intGrade3, NuberStrdents As Integer
intGrade1 = 60
intGrade2 = 70
intGrade3 = 80
NumberStudent = 3
MsgBox ("Average grade = "&
Str (intGrade1 + intGrade2 + intGrade3/ NumberStudents))
End Sub
```

When you run the program, however, it calmly informs you that the average score is 156.66666667.

That doesn't look so good, what is wrong?

The problem lies in this line:

```
Str(intGrade1 + intGrade2 + intGrade3 / NumberStudents))
```

Visual Basic evaluates the expression in parentheses from left to right, using pairs of operands and their associated operator, so it adds the first two grades together first. Instead of adding the final grade, however, it first divides that grade by

**NumberStudents**, because the division operation has higher precedence than addition.

So the result is  $60 + 70 + (80/3) = 156.66666667$ .

The solution here is to group the values to add together this way using parentheses:

```
Private Sub Command1_Click()
Dim intGrade1, intGrade2, intGrade3, NumberStudents As Integer
intGrade1 = 60
intGrade2 = 70
intGrade3 = 80
NumberStudents = 3
MsgBox ("Average grade = " &
Str((intGrade1 + intGrade2 + intGrade3)/ NumberStudents))
End Sub
```

Running this new code gives us an average of 70, as it should be.

This example points out the need to understand how Visual Basic evaluates expressions involving operators. In general, such expressions are evaluated left to right, and when it comes to a contest between two operators (such as + and / in the last term of our original program), the operator with the higher precedence is used first.

Visual Basics operator precedence, arranged by category.

Operators and

Operato

Precedence

**Comparison Logical**

**Arithmetic**

Exponentiation

(^) Equality (=)

Not

Negation (-)

Inequality

And

(<>)

Multiplication

and division

Less than

Or

(\*,/)

(<)

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Also, comparison operators actually all have equal precedence, which means they are evaluated in the left-to-right order in which they appear.

If in doubt, use parentheses operations within parentheses are always performed before those outside. Within parentheses, however, operator precedence is maintained.

### 5.2.6. Using If & Else Statements

The **If** statement is the bread and butter of Visual Basic conditionals, but you can forget the syntax every now and then (that is, is it **ElseIf** or **Else If**?), so here's the **If** statement:

```
If condition Then
[statements]
[ElseIf condition-n Then
[elseifstatements]]...
[else
[elsestatements]]
end If
```

And here's an example showing how to use the various parts of this popular statement:

```
Dim intInput
intInput = -1
While intInput < 0
```

```
intInput = InputBox("Enter a positive number")
Wend

If intInput = 1 Then
MsgBox ("Thank you".)
ElseIf intInput = 2 Then
MsgBox ("That's fine".)
ElseIf intInput >= 3 Then
MsgBox ("Too big".)
End If
```

### 5.2.7 Using Select Case

You have to get a value from the user and respond in several different ways, but you don't look forward to a long and tangled series of **If&Then&Else** statements. What can you do?

If your program can handle multiple values of a particular variable and you don't want to stack up a lot of **If&Else** statements to handle them, you should consider **Select Case**. You use **Select Case** to test an expression, seeing which of several *cases* it matches, and execute the corresponding code. Here is the syntax:

```
Select Case testexpression
[Case expressionlist-n
[statements-n]] ...
[Case Else
[elsestatements]]
End Select
```

Here is an example using **Select Case**. In this example, we read a positive value from the user and test it, responding according to its value. Note that we also use the **Select Case Is** keyword (not the same as the **Is** operator) to check if the value we read in is greater than a certain value, and **Case Else** to handle values we don't explicitly provide code for. Here is the example:

```
Dim intInput
intInput = -1
While intInput < 0
intInput = InputBox("Enter a positive number")
Wend
Const intMax = 100
Select Case intInput
Case 1:
 MsgBox ("Thank you".)
Case 2:
 MsgBox ("That's fine".)
```

Case 3:

```
MsgBox ("Your input is getting pretty big now...")
```

Case 4 to 10:

```
MsgBox ("You are approaching the maximum!")
```

Case Is > intMax:

```
MsgBox ("Too big, sorry.")
```

Case Else:

```
MsgBox ("Please try again.")
```

End Select

### 5.2.8 Making Selections With Switch() And Choose()

For some reason, few books on Visual Basic cover the **Switch()** and **Choose()** functions. They certainly have their uses, however, and we'll take a look at them here.

**The Switch() Function** The **Switch()** function evaluates a list of expressions and returns a **Variant** value or an expression associated with the first expression in the list that is true. Here is the syntax:

```
Switch (expr-1, value-1[, expr-2, value-2 ... [, expr-n, value-n]])
```

In this case, *expr-1* is the first expression to evaluate; if true, **Switch()** returns *value-1*. If *expr-1* is not True but *expr-2* is, **Switch()** returns *value-2* and so on.

Here is an example showing how to use **Switch()**. In this case, we ask the user to enter a number and use **Switch()** to calculate the absolute value of that value (having temporarily forgotten how to use the built-in Visual Basic absolute value function, **Abs()**):

```
Dim intValue
intValue = InputBox("Enter a number")
intAbsValue = Switch(intValue < 0, -1 * intValue,
intValue >= 0, intValue)
MsgBox "Absolute value = " & Str(intAbsValue)
```

### 5.2.9 The Choose() Function

You use the **Choose()** function to return one of a number of choices based on an index. Here is the syntax:

```
Choose (index, choice-1 [, choice-2, ... [, choice-n]])
```

If the index value is 1, the first choice is returned, if index equals 2, the second choice is returned, and so on.

Here is an example using **Choose()**. In this case, we have three employees Bob, Denise, and Ted with employee IDs 1, 2, and 3. This code snippet accepts an ID value from the user and uses **Choose()** to display the corresponding employee name:

```
Dim intID
intID = -1
```

```
While intID < 1 Or intID > 3
intID = InputBox("Enter employee's ID")
Wend
MsgBox "Employee name = " & Choose(intID, "Bob", "Denise", "Ted")
```

### 5.2.10 Sending Keystrokes to Other Programs

It is time to print out the 349 screen spreadsheets you have created in your new spreadsheet program to show the boss. Regrettably, there just doesn't seem to be any way to print them out except one at a time, using the File menus Print item. Can Visual Basic help here?

Yes. You can use the **SendKeys()** function to send keys to the program that currently has the Windows focus, just as if you typed in those keys yourself. Using the Alt key, you can reach the menu items in your spreadsheet's File menu. The day is saved, because now you can automate your printing job, even waiting until the spreadsheet program processes the current keystroke before continuing. Here is how you use **SendKeys()**:

**SendKeys** *string* [, *wait*]

The *string* expression is the string you want to send to the other program. The *wait* argument is a Boolean value indicating the wait mode. If False (which is the default), control returns right after the keys are sent. If True, the keystrokes must be processed by the other program before control returns.

If the keys you want to send are not simple text, just embed the codes, in the text you send to **SendKeys()**.

Send Keys ()

Key Codes,

| Key           | Code                         |
|---------------|------------------------------|
| Backspace     | {BACKSPACE}. {BS}. or {BKSP} |
| Break         | {BREAK}                      |
| Caps Lock     | {CAPSLOCK}                   |
| Del or Delete | {DELETE} or {DEL}            |
| Down arrow    | {DOWN}                       |
| End           | {END}                        |
| Enter or      | {ENTER} or                   |
| Return        |                              |
| Esc           | {ESC}                        |
| Help          | {HELP}                       |
| Home          | {HOME}                       |
| Ins or Insert | {INSERT} or {INS}            |
| Left arrow    | {LEFT}                       |



|              |              |
|--------------|--------------|
| Num Lock     | {NUMLOCK}    |
| Page Down    | {PGDN}       |
| Page Up      | {PGUP}       |
| Print Screen | {PRTSC}      |
| Righ arrow   | {RIGHT}      |
| Scroll Lock  | {SCROLLLOCK} |
| Tab          | {TAB}        |
| Up arrow     | {UP}         |
| F1           | {F1}         |
| F2           | {F2}         |
| F3           | {F3}         |
| F4           | {F4}         |
| F5           | {F5}         |
| F6           | {F6}         |
| F7           | {F7}         |
| F8           | {F8}         |
| F9           | {F9}         |
| F10          | {F10}        |
| F11          | {F11}        |
| F12          | {F12}        |
| F13          | {F13}        |
| F14          | {F14}        |
| F15          | {F15}        |
| F16          | {F16}        |
| Shift        | +            |
| Ctrl         | ^            |
| Alt          | %            |



Figure 5.1 Sending Keystrokes to Windows WordPad.

Here is an example showing how to use **SendKeys()**. Here, we give the Windows WordPad program the focus with the Visual Basic **AppActivate()** function, passing it the title of that program (which appears in its title bar), and send the string—Hello from Visual Basic! to that program as follows:

```
AppActivate ("Document - WordPad")
```

```
SendKeys ("Hello from Visual Basic!")
```

The result appears in Figure 3.1\_now we are able to send keystrokes to another program.

### 5.2.11 Handling Higher Math

Well, it may have been a mistake taking on that programming job from the astrophysics department. How do you calculate a hyperbolic cosecant anyway? Can Visual Basic do it?

Yes, although not directly. The built-in Visual Basic math functions.

|                                                      |                        |
|------------------------------------------------------|------------------------|
| Visual<br>Basic Math<br>functions<br><b>Function</b> | <b>Calculates This</b> |
| ABs                                                  | Absolute value         |
| Atn                                                  | Arc tangent            |

|     |                |
|-----|----------------|
| Cos | Cosine         |
| Exp | Exponentiation |
| Fix | Fix places     |
| Int | Integer value  |
| Log | Log            |
| Rnd | Random number  |
| Sgn | Sign           |
| Sin | Sine           |
| Sqr | Square root    |
| Tan | Tangent        |

If what you want, like hyperbolic cosecant, which shows you how to calculate other results using the built-in Visual Basic functions.

Calculated

math

functions

Calculate This Way

Function

|                   |                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------|
| Secant            | $\text{Sec}(X) = 1 / \text{Cos}(X)$                                                                      |
| Cosecant          | $\text{Cosec}(X) = 1 / \text{Sin}(X)$                                                                    |
| Cotangent         | $\text{Cotan}(X) = 1 / \text{Tan}(X)$                                                                    |
| Inverse sine      | $\text{Arcsin}(X) = \text{Atn}(X / \text{Sqr}(-X * X + 1))$                                              |
| Inverse cosine    | $\text{Arccos}(X) = \text{Atn}(-X / \text{Sqr}(-X * X + 1)) + 2 * \text{Atn}(1)$                         |
| Inverse secant    | $\text{Arccosec}(X) = \text{Atn}(X / \text{Sqr}(X * X - 1)) + \text{Sgn}((X - 1) * (2 * \text{Atn}(1)))$ |
| Inverse cosecant  | $\text{Arccosec}(X) = \text{Atn}(X / \text{Sqr}(X * X - 1)) + (\text{Sgn}(X) - 1) * (2 * \text{Atn}(1))$ |
| Inverse cotangent | $\text{Arccotan}(X) = \text{Atn}(X) + 2 * \text{Atn}(1)$                                                 |
| Inverse cotangent | $\text{HSin}(X) = (\text{Exp}(X) - \text{Exp}(-X)) / 2$                                                  |
| Hyperbolic sine   | $\text{HCos}(X) = (\text{Exp}(X) + \text{Exp}(-X)) / 2$                                                  |
| Hyperbolic cosine | $\text{HTan}(X) = (\text{Exp}(X) - \text{Exp}(-X)) / (\text{Exp}(X) + \text{Exp}(-X))$                   |
|                   | $\text{HSec}(X) = 2 / (\text{Exp}(X) + \text{Exp}(-X))$                                                  |

|            |                                                                                        |
|------------|----------------------------------------------------------------------------------------|
| Hyperbolic | $\text{HCoSec}(X) = 2/(\text{Exp}(X) - \text{Exp}(-X))$                                |
| tangent    |                                                                                        |
| Hyperbolic | $\text{HCotan}(X) = (\text{Exp}(X) + \text{Exp}(-X))/(\text{Exp}(X) - \text{Exp}(-X))$ |
| secant     |                                                                                        |
| Hyperbolic | $\text{HArccsin}(X) = \text{Log}(X + \text{Sqr}(X^2 + 1))$                             |
| cosecant   |                                                                                        |
| Hyperbolic | $\text{HArcccos}(X) = \text{Log}(X + \text{Sqr}(X^2 - 1))$                             |
| cotangent  |                                                                                        |
| Inverse    |                                                                                        |
| hyperbolic |                                                                                        |
| sine       |                                                                                        |
| Inverse    |                                                                                        |
| hyperbolic |                                                                                        |
| cosine     |                                                                                        |

### 5.2.12 Handling Dates and Times

One of the biggest headaches a programmer can have is working with dates. Handling hours, minutes, and seconds can be as bad as working with pounds, shillings, and pence. Fortunately, Visual Basic has a number of date and time handling functions.

Visual Basic

date

keywords. Use This

To do This

|               |                             |
|---------------|-----------------------------|
| Get the       | Date, Now, Time             |
| Current date  |                             |
| or time       |                             |
| Perform date  | DateAdd, DateDiff, DatePart |
| calculations  |                             |
| Return a data | DateSerial, DateValue       |
| Return a time | TimeSerial, TimeValue       |
| Set the data  | Date, Time                  |
| or time       |                             |
| Time a        | Timer                       |
| process       |                             |

There is something else you should know the Format\$() function makes it easy to format dates into strings, including times. Which shows how to display the date and time in a string note how many ways there are to do this?

FormatS () to

display dates and

times. Format Yields This on january I, 2000 at 1:00 A.M.

Expression

|                   |                        |
|-------------------|------------------------|
| Format \$ (Now, m |                        |
| -d-yy)            | 1-1-00                 |
| Format\$ (Now, m  |                        |
| /d/yy)            | 1/1/00                 |
| Format\$ (Now,    |                        |
| mm-dd-yy)         | 01-01-00               |
| Format\$ (Now,    |                        |
| ddd, mmmm d,      | Friday January 1, 2000 |
| yyy)              |                        |
| Format\$ (Now, d  | 1 Jan. 2000            |
| mmm,yyy           |                        |
| Format\$ (Now,    |                        |
| hh:mm:ss          | 01:00:00 01/01/00      |
| mm/dd/yy)         |                        |
| Format\$ (Now,    |                        |
| hh:mm:ss          | 01:00:00 AM 01-01-00   |
| AM/PM mm-dd-      |                        |
| yy)               |                        |

You can also compare dates and times directly. For example, here\_s how you loop until the current time (returned as a string by **Time\$**) exceeds the time the user has entered in a text box (for example, 15:00:00); when the time is up, the program beeps and displays

a message box:

```
While Time$ < Text1.Text
Wend
Beep
MsgBox ("Time_s up!")
```



Caution

Do not use this code snippet for more than an example of how to compare times! The eternal looping while waiting for something to happen is a bad idea in Windows, because your program monopolizes a lot of resources that way. Instead, set up a Visual Basic Timer object and have a procedure called, say, every second.

### **5.3 Concatenation Operators**

Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and &. Both carry out the basic concatenation operation, as the following example shows.

```
Dim x As String = "Con" & "caten" & "ation"
Dim y As String = "Con" + "caten" + "ation"
'The preceding statements set both x and y to "Concatenation".
```

These operators can also concatenate **String** variables, as the following example shows.

```
Dim a As String = "abc"
Dim d As String = "def"
Dim z As String = a & d
Dim w As String = a + d
' The preceding statements set both z and w to "abcdef".
```

#### **5.3.1 Differences Between the Two Concatenation Operators**

The + Operator (Visual Basic) has the primary purpose of adding two numbers. However, it can also concatenate numeric operands with string operands. The + operator has a complex set of rules that determine whether to add, concatenate, signal a compiler error, or throw a run-time Invalid Cast Exception exception.

The & Operator (Visual Basic) is defined only for String operands, and it always widens its operands to String, regardless of the setting of Option Strict. The & operator is recommended for string concatenation because it is defined exclusively for strings and reduces your chances of generating an unintended conversion.

### **5.4 Logical Operator**

The logical operators compare Boolean expressions and return a Boolean result. The And, Or, AndAlso, OrElse, and Xor operators take two operands, and the Not operator takes a single operand.

The Not operator performs logical negation on a **Boolean** expression. Put simply, it yields the opposite of the expression it evaluates. If the expression evaluates to True, Not yields False; if the expression evaluates to False, Not yields True. An example is given on next page.

```
Dim x As Boolean
x = Not 23 > 12 ' x equals False
x = Not 23 > 67 ' x equals True.
```

The **And** operator performs logical conjunction on two **Boolean** expressions. That is, if both expressions evaluate to **True**, then the And operator returns **True**. If either or both expressions evaluate to **False**, then **And** returns **False**.

The **Or** operator performs logical disjunction on two **Boolean** expressions. If either expression evaluates to **True**, **Or** returns **True**. If neither expression evaluates to **True**, **Or** returns **False**.

**Xor** performs logical exclusion on two expressions. If either expression evaluates to **True**, but not both, **Xor** returns **True**. If both expressions evaluate to **True** or both expressions evaluate to **False**, **Xor** returns **False**.

Examples of the **And**, **Or**, and **Xor** operators are shown below:

```
Dim x As Boolean
x = 23 > 12 And 12 > 4 ' x = True
x = 12 > 23 And 12 > 4 ' x = False
x = 23 > 12 Or 4 > 12 ' x = True
x = 23 > 45 Or 4 > 12 ' x = False
x = 23 > 45 Xor 12 > 4 ' x = True
x = 23 > 12 Xor 12 > 4 ' x = False
x = 12 > 23 Xor 4 > 12 ' x = False
```



Notes

In addition to being logical operators, **Not**, **Or**, **And**, and **Xor** also perform bitwise arithmetic when used on numeric values. Information on this functionality can be found at Arithmetic Operators.

The **AndAlso** operator is very similar to the **And** operator, in that it also performs logical conjunction on two **Boolean** expressions. The key difference between **AndAlso** and **And** is that **AndAlso** exhibits short-circuiting behavior. If the first expression in an **AndAlso** expression evaluates to **False**, then the second expression is not evaluated and **False** is returned for the **AndAlso** expression.

Similarly, the **OrElse** operator performs short-circuiting logical disjunction on two **Boolean** expressions. If the first expression in an **OrElse** expression evaluates to **True**, then the second expression is not evaluated and **True** is returned for the **OrElse** expression. Below are some examples that illustrate the difference between **And**, **Or**, and their counterparts:

```
12 > 45 And MyFunction(4) ' MyFunction() is called.
12 > 45 AndAlso MyFunction(4) ' MyFunction() is not called.
45 > 12 Or MyFunction(4) ' MyFunction is called.
45 > 12 OrElse MyFunction(4) ' MyFunction is not called
```

In the first example, MyFunction() is called, even though  $12 > 45$  returns **False**, because **And** does not short circuit. In the second example, MyFunction is not called, because  $12 > 45$  returns **False**, so **AndAlso** short circuits the second expression. In the third example, MyFunction is called, even though  $45 > 12$  returns **True**, because **Or** does not short circuit. In the fourth example, MyFunction is not called because  $45 > 12$  returns **True**, so **OrElse** short circuits the second expression



#### Case Study

Consider the following program that displays the address of the Delhi House. (The form design for all examples in this section consists of a button and a text or list box.)

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 Dim houseNumber As Double
 Dim street, address As String
 houseNumber = 1600
 street = "Pennsylvania Ave."
 address = houseNumber & " " & street
 txtAddr.Text = "The Delhi House is located at " & address
End Sub
```

[Run, and then click the button. The following is displayed in the text box.]

The Delhi House is located at Safadarjung.

#### Questions

1. Compare the three number using operators.
2. How to convert string into integers?

## 5.5 Summary

- String are the most common type of data processed by visual basic.
- Math operators are addition and subtraction.
- Visual basic evaluates the expression in parentheses from left to right, using pairs of operands and their associated operator.
- The logical operators compare boolean expressions and return a boolean result.

## 5.6 Keywords

**Concatenation operators:** Join multiple strings in to a single string. They are two operators, t and f.

**Exponents:** They are known as powers of a number. They are used is many things which are represented as power of two.



**Fixed length strings:** Fixed length strings are automatically filled with spaces to pad them to their fixed-length.

**Logical operators:** It compares Boolean expressions and return a Boolean result.

**Path compact path ex:** It is the converse of the size string. It is used to shorten a string to a specified length. It is intended to trim fully qualified filenames to a specified number of characters, etc.



Lab Exercise

1. How can I find the strings after compilation?
2. How do I search for files and replace strings within them?

### 5.7 Self Assessment Questions

1. Strings are automatically filled with spaces to pad them to their fixed length
  - (a) Constant \_ length
  - (b) Fixed-length
  - (c) Variable-length
  - (d) None of the above
2. The API Function will put a \_\_\_\_\_ character at the end of the actual dath in the string variable.
  - (a) Null
  - (b) Zero
  - (c) Two
  - (d) All of the above
3. Integer division divides one number into another, and then velirms the inliger portion of the result.
  - (a) True
  - (b) False
4. Function is used to sent keys to the programme that currently has the windows focuss.
  - (a) AppActive ()
  - (b) Sendkeys ()
  - (c) Choose ()
  - (d) Switch ()
5. Not operator performs logical negation on a Boolean expression.
  - (a) True
  - (b) False

### 5.8 Review Questions

1. Define the logical operator.
2. How many positions does a string of eight characters have?
3. What is the highest numbered position for a string of eight characters?
4. What is the difference between a database and data structure?
5. Can I send keystrokes to a DOS application?
6. How to create a form with resizing borders and no title bar?

### **Answers for Self Assessment Questions**

1. (b)
2. (a)
3. (a)
4. (b)
5. (a)

### **5.9 Further Readings**



*Books*

G. Cornell, "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Milan, "Visual Basic .Net".



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## Unit 6: VB Control Structure

### CONTENTS

Objectives

Introduction

6.1 Structure

6.2 If-else Blocks

6.2.1 If Block

6.2.2 Elself Clauses

6.3 Do While Loops

6.4 While-Wend

6.5 For... Next Loops

6.5.1 Declaration of Control Variables

6.5.2 Nested For... Next Loops

6.6 Select Case Blocks

6.7 Summary

6.8 Keywords

6.9 Self Assessment Questions

6.10 Review Questions

6.11 Further Readings

### Objectives

*After studying this unit, you will be able to:*

- Understand the structure of VB control and if else statement
- Discuss about the do while loops and next loops
- Explain about the case blocks.

### Introduction

We have seen many examples of the usefulness of subscripted variables. They are essential for writing concise solutions to many programming problems. Because of the utility that subscripts provide, Visual Basic also provides for array of labels, text boxes, and buttons. Since labels, text boxes, and buttons are referred to generically in Visual Basic as control, array of these objects (or of any other control ) are called.



Notes

Since Visual Basic does not provide for array of masked text boxes, we will only use ordinary text boxes for input.

Control array are created in much the same way as any other array; that is, with a statement of the form

```
Dim arrayName(n) As ControlType
```

or

```
Dim arrayName() As ControlType
```

For instance, the following statements declare control array.

```
Dim lblTitle(10) As Label
```

```
Dim txtNumber(8) As TextBox
```

```
Dim btnAmount() As Button
```



*Example 1:* A department store has five departments. The following program requests the amount of sales for each department and displays the total sales for the store. The five labels identifying the departments are grouped into an array of labels and the five text boxes for the individual amounts are grouped into an array of text boxes. The initial settings for these labels are specified at run time in the frmSales\_Load event procedure instead of being specified at design time. The Text properties of the labels are set to be "Department 1", "Department 2", and so on.

| Object                                   | Property | Setting             |
|------------------------------------------|----------|---------------------|
| frmSales                                 | Text     | Daily Sales:        |
| Label1Label5 TextBox1TextBox5 btnCompute | Text     | Compute Total Sales |
| lblTotal                                 | Text     | Total Sales         |
| txtTotal                                 | ReadOnly | True                |
| Dim lblDept(4) As Label                  |          |                     |
| Dim txtDept(4) As TextBox                |          |                     |

```
Private Sub frmSales_Load(...) Handles MyBase. Load
 lblDept(0) = Label1
 lblDept(1) = Label2
 lblDept(2) = Label3
 lblDept(3) = Label4
 lblDept(4) = Label5
 txtDept(0) = TextBox1
 txtDept(1) = TextBox2
 txtDept(2) = TextBox3
 txtDept(3) = TextBox4
 txtDept(4) = TextBox5
 'Set the labels' Text property to the corresponding department
 'Set the text boxes' Text property to the empty string
 For depNum As Integer = 1 To 5
 lblDept(depNum - 1).Text = "Department " & depNum & ":"
 Next
End Sub

Private Sub btnCompute_Click(...) Handles btnCompute.Click
 Dim totalSales As Double = 0
 For depNum As Integer = 1 To 5
 totalSales += Cdbl(txtDept(depNum - 1).Text)
 Next
 txtTotal.Text = FormatCurrency(totalSales)
End Sub
```

[Run, enter amounts into the text boxes, and then click the button.]



| Department          | Sales Amount       |
|---------------------|--------------------|
| Department 1:       | 2114.35            |
| Department 2:       | 1843.28            |
| Department 3:       | 3662.00            |
| Department 4:       | 1183.43            |
| Department 5:       | 1955.02            |
| <b>Total Sales:</b> | <b>\$10,758.08</b> |

## 6.1 Structure

Some of the data types we have worked with so far are Double, Integer, String, and Boolean. In previous versions of Visual Basic a Structure was called a user-defined type (UDT). Three pieces of related information about colleges are "name," "state," and "year founded." A Structure type capable of holding this data is defined by the following block of statements located in the Declarations section of the Code window.

```

College
Dim name As String
Dim state As String
Dim yearFounded As Integer
End

```

Each of the three subvariables of the Structure type is called a member. A Structure variable of the type College is declared by a statement such as Dim college1 As College

Each member is accessed by giving the name of the Structure variable and the member, separated by a period. For instance, the three members of the Structure variable college1 are accessed as college1.name, college1.state, an college1.yearFounded. In general, a Structure type is defined by a Structure block of the form

```

StructureType
Dim memberName1 As MemberType1
Dim membername2 As MemberType2
.
.
End

```

where StructureType is the name of the user-defined data type; memberName1, memberName2,..., are the names of the members of the Structure; and MemberType1, MemberType2,..., are the corresponding member types, such as String, Integer, Double, or Boolean.

A Structure variable, structureVar, is declared to be of the user-defined type by a statement of the form Dim structureVar As StructureType.

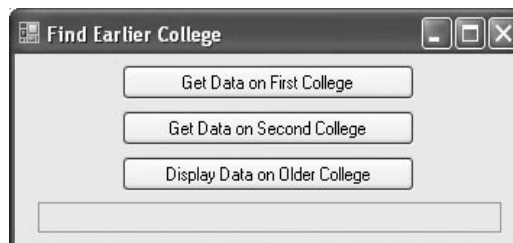


Did u know?

Subvariables of the Structure type is called a member.



*Example 1:* The following program stores information about two colleges into Structure variables of type College and uses the variables to determine which college is older. Note: In the third line of the btnOlder\_Click event procedure, all the member values of college1 are assigned simultaneously to collegeOlder.



| Object    | Property | Setting                       |
|-----------|----------|-------------------------------|
| frmFirst  | Text     | Find Earlier College          |
| btnFirst  | Text     | Get Data on First College     |
| btnSecond | Text     | Get Data on Second College    |
| btnOlder  | Text     | Display Data on Older College |
| txtResult | ReadOnly | True                          |
| College   |          |                               |

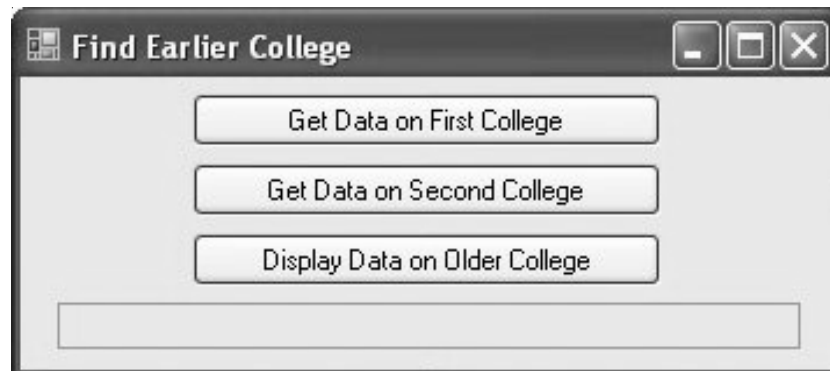
```

Dim name As String
Dim state As String
Dim yearFounded As Integer
End Structure

Dim college1, college2, collegeOlder As College
Private Sub btnFirst_Click(...) Handles btnFirst.Click
 Dim prompt As String
 college1.name = InputBox("Enter name of first college.", "Name")
 college1.state = InputBox("Enter state of first college.", "State")
 prompt = "Enter the year the first college was founded."
 college1.yearFounded = CInt(InputBox(prompt, "Year"))
End Sub
Private Sub btnSecond_Click(...) Handles btnSecond.Click
 Dim prompt As String
 college2.name = InputBox("Enter name of second college.", "Name")
 college2.state = InputBox("Enter state of second college.", "State")
 prompt = "Enter the year the second college was founded."
 college2.yearFounded = CInt(InputBox(prompt, "Year"))
End Sub
Private Sub btnOlder_Click(...) Handles btnOlder.Click
 If (college1.yearFounded < college2.yearFounded) Then
 collegeOlder = college1
 Else
 collegeOlder = college2
 End If
 txtResult.Text = collegeOlder.name & " was founded in " & _
 collegeOlder.state & " in " & collegeOlder.yearFounded & "."
End Sub

[Run, click the first button, and respond to the input boxes with Delhi University, , 1746. Click
the second button; respond to the input boxes with NIPS University, Connecticut, 1701. Finally,
click the third button.]

```



## 6.2 If- else Blocks

An If block allows a program to decide on a course of action based on whether certain conditions are true or false.

### 6.2.1 If Block

A block of the form:

If condition Then

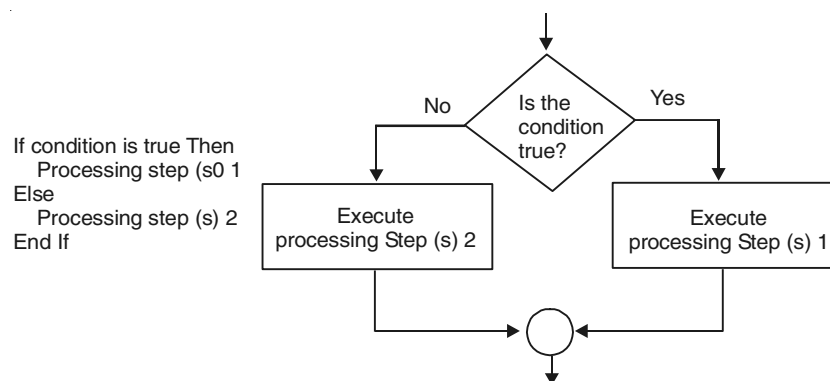
    action1

else

    action2

end If

causes the program to take action1 If condition is true and action2 If condition is false. Each action consists of one or more Visual Basic statements. After an action is taken, execution continues with the line after the If block. Figure 6.1 contains the pseudocode and flowchart for an If block.



**Figure 6.1:** Pseudocode and Flowchart for an If Block.



*Example 1:* The following program finds the larger of two numbers input by the user. The condition is

num1 > num2



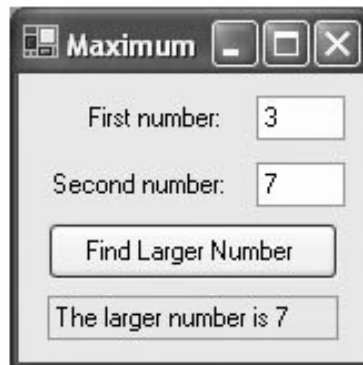
and each action consists of a single assignment statement. With the inputs 3 and 7, the condition is false, and so the second action is taken.



| Object        | Property | Setting            |
|---------------|----------|--------------------|
| FrmMaximum    | Text     | Maximum            |
| LblFirstNum   | Text     | First Number:      |
| TxtFirstNum   |          |                    |
| LblSecondNum  | Text     | Second Number:     |
| TxtSecondNum  |          |                    |
| BtnFindLarger | Text     | Find Larger Number |
| TxtResult     | ReadOnly | True               |

```
Private Sub btnFindLarger_Click(...) Handles btnFindLarger.Click
 Dim num1, num2, largerNum As Double
 num1 = Cdbl (txtFirstNum.Text)
 num2 = Cdbl (txtSecondNum.Text)
 If num1 > num2 Then
 largerNum = num1
 Else
 largerNum = num2
 EndIf
 txtResult.Text = "The larger number is " & largerNum
End Sub
```

[Run, type 3 and 7 into the text boxes, and press the button.]



*Example 2:* The following program requests the costs and revenue for a company and displays the message “Break even” if the costs and revenue are equal; otherwise, it displays the profit or loss. The action following Else is another If block.



| Object    | Property  | Setting               |
|-----------|-----------|-----------------------|
| frmStatus | Text      | Profit/Loss           |
| lblCosts  | Text      | Costs:                |
| txtCosts  |           |                       |
| lblRev    | Text      | Revenue:              |
| txtRev    |           |                       |
| btnShow   | Text      | Show Financial Status |
| txtResult | Read Only | True                  |

```

Private Sub btnShow_Click(...) Handles btnShow.Click
 Dim costs, revenue, profit, loss As Double
 costs = Cdbl (txtCosts.Text)
 revenue = Cdbl (txtRev.Text)
 If costs = revenue Then

```

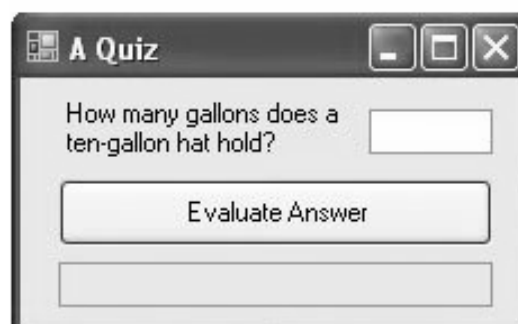
```

txtResult.Text = "Break even"
Else
 If costs < revenue Then
 profit = revenue - costs
 txtResult.Text = "Profit is " & FormatCurrency(profit)
 Else
 loss = costs - revenue
 txtResult.Text = "Loss is " & FormatCurrency(loss)
 End If
End If
End Sub
[Run, type 9500 and 8000 into the text boxes, and press the button.]

```



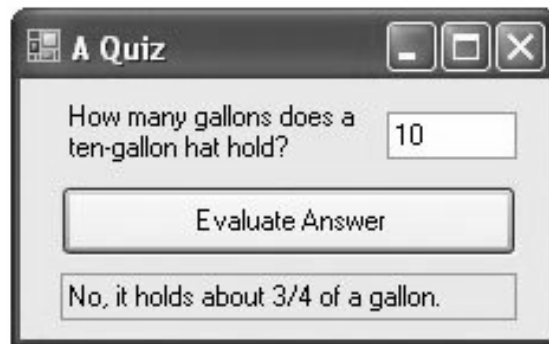
*Example 3:* The If block in the following program has a logical operator in its condition.



| Object      | Property | Setting                                        |
|-------------|----------|------------------------------------------------|
| frmQuiz     | Text     | A Quiz                                         |
| lblQuestion | Text     | How many gallons does a<br>tengallon hat hold? |
| btnEvaluate | Text     | Evaluate Answer                                |
| txtSolution | ReadOnly | True                                           |

```
Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click
 'Evaluate answer
 Dim answer As Double
 answer = CDb1 (txtAnswer.Text)
 If (answer >= 0.5) And (answer <= 1) Then
 txtSolution.Text = "Good, "
 Else
 txtSolution.Text = "No, "
 End If
 txtSolution.Text &= "it holds about 3/4 of a gallon."
End Sub
```

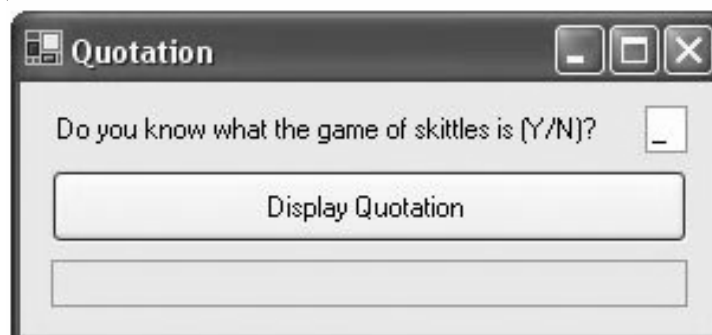
[Run, type 10 into the text box, and press the button.]



The Else part of an If block can be omitted. This important type of If block appears in the next example.



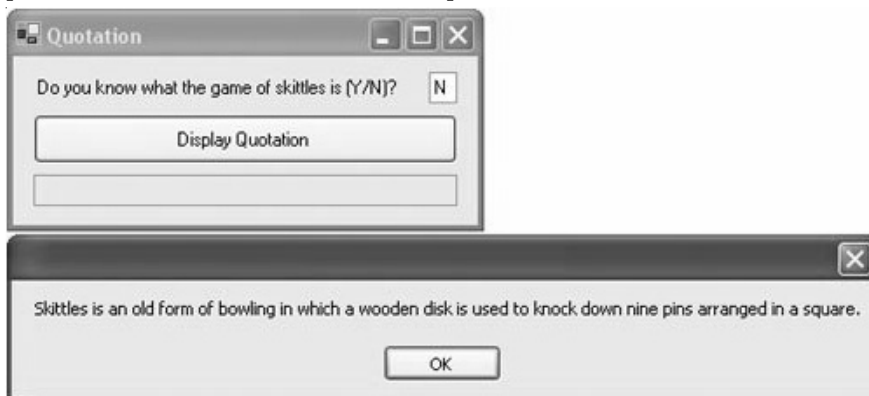
*Example 4:* The following program offers assistance to the user before presenting a quotation.



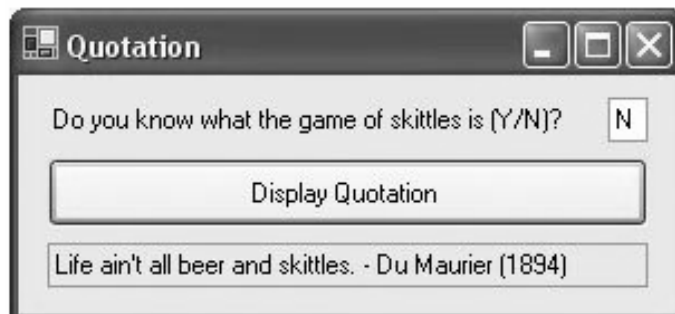
| Object       | Property | Setting                                         |
|--------------|----------|-------------------------------------------------|
| frmQuotation | Text     | Quotation                                       |
| LblQuestion  | Text     | Do you know what the game of skittles is (Y/N)? |
| mtxtAnswer   | Mask     | L                                               |
| BtnDisplay   | Text     | Display Quotation                               |
| TxtQuote     | ReadOnly | True                                            |

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 Dim message As String
 message = "Skittles is an old form of bowling in which a wooden " & _
 "disk is used to knock down nine pins arranged in a
square."
 If mtxtAnswer.Text.ToUpper = "N" Then
 MsgBox(message, 0, "")
 End If
 txtQuote.Text = "Life ain't all beer and skittles." & _
 " - Du Maurier (1894)"
End Sub
```

[Run, type "N" into the masked text box, and press the button.]



[Press OK.]





Notes

Rerun the program, type "Y" into the masked text box, press the button, and observe that the description of the game is skipped.

### 6.2.2 ElseIf Clauses

An extension of the If block allows for more than two possible alternatives with the inclusion of ElseIf clauses. A typical block of this type is

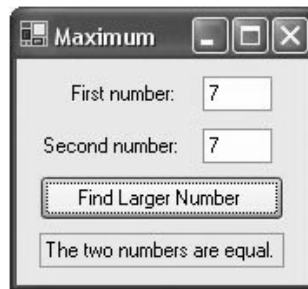
```
If condition1 Then
 action1
ElseIf condition2 Then
 action2
ElseIf condition3 Then
 action3
Else
 action4
End If
```

This block searches for the first true condition, carries out its action, and then skips to the statement following End If. If none of the conditions is true, then Else's action is carried out.

Execution then continues with the statement following the block. In general, an If block can contain any number of elseif clauses. As before, the Else clause is optional.



*Example 5:* The following program redoes Example 5 so that if the two numbers are equal, the program so reports:



```
Private Sub btnFindLarger_Click(...) Handles btnFindLarger.Click
 Dim num1, num2 As Double
 num1 = Cdbl (txtFirstNum.Text)
 num2 = Cdbl (txtSecondNum.Text)
 If (num1 > num2) Then
 txtResult.Text = "The larger number is " & num1
 ElseIf (num2 > num1) Then
 txtResult.Text = "The larger number is " & num2
 Else
 txtResult.Text = "The two numbers are equal."
```

```

End If
End Sub
[Run, type 7 into both text boxes, and press the button.]

```

If blocks allow us to define functions whose values are not determined by a simple formula. The function in [10](#) uses an If block.



*Example 6:* The Social Security or FICA tax has two components the Social Security benefits tax, which in 2005 is 6.2 percent on the first \$90,000 of earnings for the year, and the Medicare tax, which is 1.45 percent of earnings. The following program calculates an employee's FICA tax for the current pay period.

| Object       | Property | Setting                                                       |
|--------------|----------|---------------------------------------------------------------|
| frm FICA     | Text     | FICA Taxes                                                    |
| lblToDate    | Text     | Total earnings for this year prior to the current pay period: |
| txtToDate    |          |                                                               |
| lblCurrent   | Text     | Earnings for the current pay period:                          |
| txtCurrent   |          |                                                               |
| btnCalculate | Text     | Calculate FICA Taxes                                          |
| lblTax       | Text     | FICA taxes for the current pay period:                        |
| txtTax       | ReadOnly | True                                                          |

```

Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
 Dim ficaTaxes As Double
 ficaTaxes = CalcFICA(CDbl (txtToDate.Text), CDbl (txtCurrent.Text))
 txtTax.Text = FormatCurrency(ficaTaxes)
End Sub
Function CalcFICA(ByVal ytdEarnings As Double, _

```

```
ByVal curEarnings As Double) As Double
'Calculate Social Security benefits tax and Medicare tax
'for a single pay period in 2005
Dim socialSecurityBenTax, medicareTax As Double
If (ytdEarnings + curEarnings) <= 90000 Then
 socialSecurityBenTax = 0.062 * curEarnings
ElseIf ytdEarnings < 90000 Then
 socialSecurityBenTax = 0.062 * (90000 - ytdEarnings)
End If
medicareTax = 0.0145 * curEarnings
Return socialSecurityBenTax + medicareTax
End Function
```

[Run, type 12345.67 and 543.21 into the top two text boxes, and press the button. The following is displayed in txtTax.]

\$41.56

### Comments

1. Constructs in which an If block is contained inside another If block are referred to as nested If blocks.
2. Some programs call for selecting among many possibilities. Although such tasks can be accomplished with complicated If blocks, the Select Case block (discussed in the next section) is often a better alternative.



Task

Write a program by using If-else block.

## 6.3 Do While Loops

A loop, one of the most important structures in Visual Basic, is used to repeat a sequence of statements a number of times. At each repetition, or pass, the statements act upon variables whose values are changing.

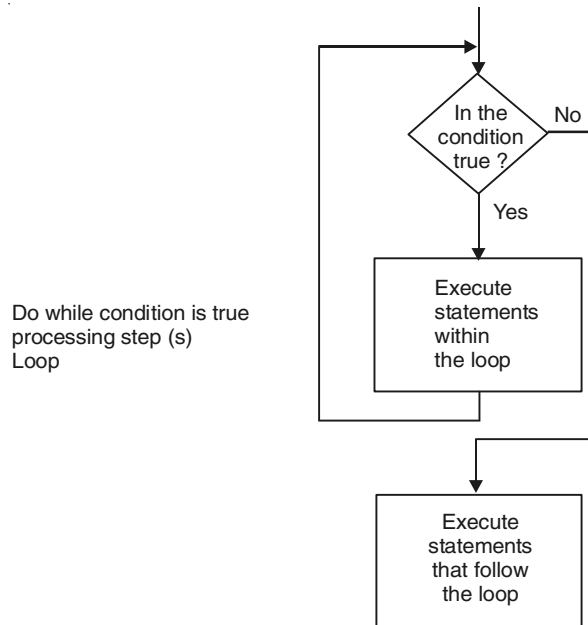
The Do loop repeats a sequence of statements either as long as or until a certain condition is true. A Do statement precedes the sequence of statements, and a Loop statement follows the sequence of statements. The condition, preceded by either the word "While" or the word "Until", follows the word "Do" or the word "Loop". When Visual Basic executes a Do loop of the form

```
Do While condition
 statement(s)
Loop
```

It first checks the truth value of condition. If condition is false, then the statements inside the loop are not executed, and the program continues with the line after the Loop statement. If condition is true, then the statements inside the loop are executed. When the statement Loop is



encountered, the entire process is repeated, beginning with the testing of condition in the Do While statement. In other words, the statements inside the loop are repeatedly executed only as long as (that is, while) the condition is true. Figure 6.2 contains the pseudocode and flowchart for this loop.



**Figure 6.2:** Pseudocode and Flowchart for a Do While Loop.



*Example 1:* The following program, in which the condition in the Do loop is “num <=7”, displays the numbers from 1 through 7. (After the Do loop executes, the value of num will be 8.)

```

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 'Display the numbers from 1 to 7
 Dim num As Integer = 1
 DoWhile num <= 7
 lstNumbers.Items.Add(num)
 num += 1 'Add 1 to the value of num
 Loop
End Sub

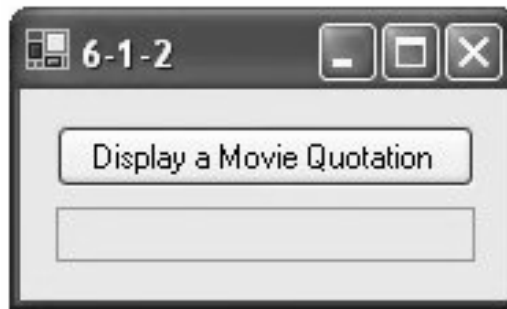
[Run, and click the button. The following is displayed in the list box.]
1
2
3
4
5
6
7

```

Do loops are commonly used to ensure that a proper response is received from the InputBox function.



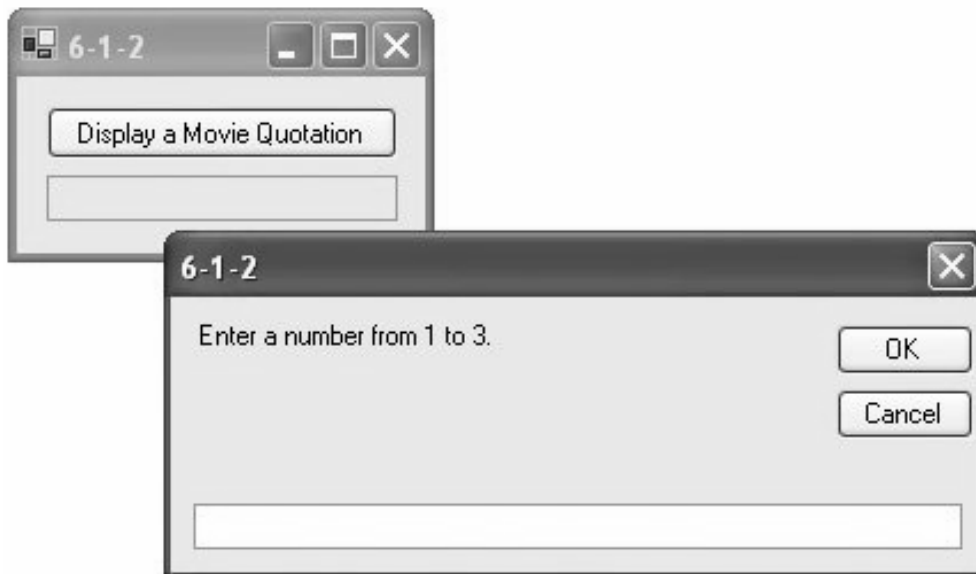
*Example 2:* The following program requires the user to enter a number from 1-3. The Do loop repeats the request until the user gives a proper response.



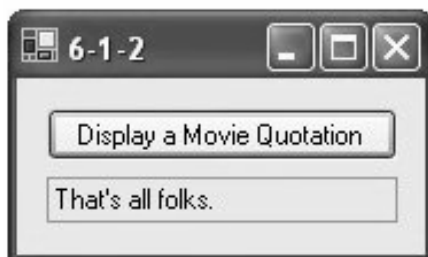
| Object       | Property | Setting                   |
|--------------|----------|---------------------------|
| frmMovie     | Text     | 6-1-2                     |
| btnDisplay   | Text     | Display a Movie Quotation |
| txtQuotation | ReadOnly | True                      |

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 Dim response As Integer, quotation As String = ""
 response = CInt(InputBox("Enter a number from 1 to 3.))
 Do While (response < 1) Or (response > 3)
 response = CInt(InputBox("Enter a number from 1 to 3.))
 Loop
 Select Case response
 Case 1
 quotation = "Plastics."
 Case 2
 quotation = "Pardip."

 Case 3
 quotation = "That's all folks."
 End Select
 txtQuotation.Text = quotation
End Sub
[Run, and click the button.]
```



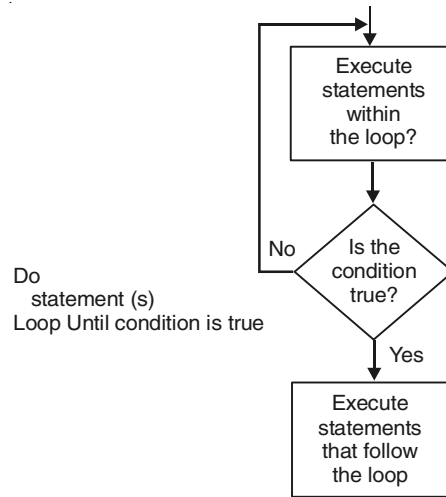
[Type 3 into the box and press the OK button.]



In Examples 1 and 2, the condition was checked at the top of the loop that is, before the statements were executed. Alternatively, the condition can be checked at the bottom of the loop when the statement Loop is reached. When Visual Basic encounters a Do loop of the form

```
Do
 statement(s)
Loop Until condition
```

it executes the statements inside the loop and then checks the truth value of condition. If condition is true, then the program continues with the line after the Loop statement. If condition is false, then the entire process is repeated beginning with the Do statement. In other words, the statements inside the loop are executed once and then are repeatedly executed until the condition is true. Figure 6.3 shows the pseudocode and flowchart for this type of Do loop.



**Figure 6.3:** Pseudocode and Flowchart for a Do loop with Condition Tested at the Bottom.



*Example 3:* The following program is equivalent to Example 12, except that the condition is tested at the bottom of the loop:

```

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 Dim response As Integer, quotation As String = ""
 Do
 response = CInt(InputBox("Enter a number from 1 to 3.))
 Loop Until (response >= 1) And (response <= 3)
 Select Case response
 Case 1
 quotation = "Plastics."
 Case 2
 quotation = "Pardip."
 Case 3
 quotation = "That's all folks."
 End Select
 txtQuotation.Text = quotation
End Sub

```

Do loops allow us to calculate useful quantities for which we might not know a simple formula.



*Example 4:* Suppose you deposit money into a savings account and let it accumulate at six percent interest compounded annually. The following program determines when you will be a millionaire:



| Object         | Property | Setting                                 |
|----------------|----------|-----------------------------------------|
| frmMillionaire | Text     | 6% Interest                             |
| LblAmount      | Text     | Amount Deposited:                       |
| TxtAmount      |          |                                         |
| btnCalculate   | Text     | Calculate Years to Become a Millionaire |
| TxtWhen        | ReadOnly | True                                    |

```
Private Sub btnYears_Click(...) Handles btnYears.Click
 'Compute years required to become a millionaire
 Dim balance As Double, numYears As Integer
 balance = CDbl(txtAmount.Text)
 DoWhile balance < 1000000
 balance += 0.06 * balance
 numYears += 1
 Loop
 txtWhen.Text = "In " & numYears & _
 " years you will have a million dollars."
End Sub
```

[Run, type 100000 into the text box, and press the button.]



## Comments

1. Be careful to avoid infinite loops that is, loops that are never exited. The following loop is infinite, because the condition “balance < 1000” will always be true.



Click on the form's Close button at the upper right corner of the title bar to end the program.

```
Private Sub btnButton_Click(...) Handles btnlick
 'An infinite loop
 Dim balance As Double = 100, intRate As Double
 Do While balance < 1000
 balance = (1 + intRate) * balance
 Loop
 txtBalance.Text = FormatCurrency(balance)
End Sub
```

Notice that this slip-up can be avoided by adding something like = 0.04 to the end of the Dim statement.

2. Visual Basic allows the use of the words “While” and “Until” at either the top or bottom of a Do loop. In this text, the usage of these words is restricted for the following reasons:
  - a. Because any While statement can be easily converted to an Until statement and vice versa, the restriction produces no loss of capabilities and the programmer has one less matter to think about.
  - b. Restricting the use simplifies reading the program. The word “While” proclaims testing at the top, and the word “Until” proclaims testing at the bottom.
  - c. Certain other major structured languages only allow “While” at the top and “Until” at the bottom of a loop. Therefore, following this convention will make life easier for people already familiar with or planning to learn one of these languages.
  - d. Standard pseudocode uses the word “While” to denote testing a loop at the top and the word “Until” to denote testing at the bottom.

## 6.4 While-Wend

The **While...Wend** statement repeats a block of code while a condition is true. The condition is always evaluated before entering the loop. If it is not fulfilled, the loop is never executed.

It is possible to exit a **While...Wend** loop at any moment using the **Exit While** statement. Control is then transferred to the first line following the **Wend** keyword. In the case of nested loops, the **Exit While** statement only exits the loop in which it is used and the external loop continues executing normally.

The following example uses a loop to list the graphics modes supported by the PDA.

```
Private Sub Button1_Click()
```

```
Dim sc as New ScreenMode
Dim b as Boolean
b = sc.FindFirstMode
While b
 MsgBox "BitDepth = " & sc.BitDepth & "\nColor = " & sc.Color
 b = sc.FindNextMode
Wend
End Sub
```

### **6.5 For... Next Loops**

When we know exactly how many times a loop should be executed, a special type of loop, called a For... Next loop, can be used. For... Next loops are easy to read and write, and have features that make them ideal for certain common tasks. The following code uses a For... Next loop to display a table:

```
Private Sub btnDisplayTable_Click(...) Handles btnDisplayTable.Click
 'Display a table of the first 5 numbers and their squares
 'Assume the font for lstTable is Courier New
 Dim i As Integer
 For i = 1 To 5
 lstTable.Items.Add(i & " " & i ^ 2)
 Next
End Sub
```

[Run, and click on btnDisplayTable. The following is displayed in the list box.]

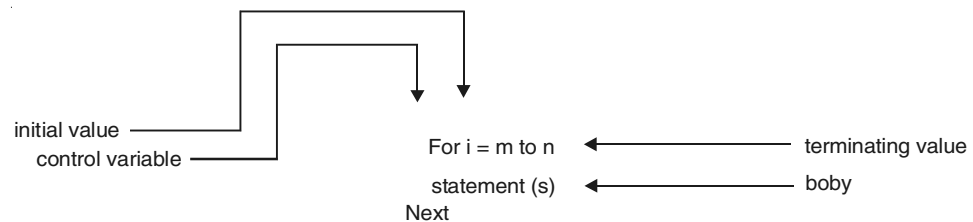
```
1 1
2 4
3 9
4 16
5 25
```

The equivalent program written with a Do loop is as follows.

```
Private Sub btnDisplayTable_Click(...) Handles btnDisplayTable.Click
 'Display a table of the first 5 numbers and their squares
 Dim i As Integer
 i = 1
 Do While i <= 5
```

```
lstTable.Items.Add(i & " " & i ^ 2)
i += 1 'Add 1 to i
Loop
End Sub
```

In general, a portion of a program of the form



constitutes a For... Next loop. The pair of statements For and Next cause the statements between them to be repeated a specified number of times. The For statement designates a numeric variable, called the control variable, that is initialized and then automatically changes after each execution of the loop. Also, the For statement gives the range of values this variable will assume. The Next statement increments the control variable. If  $m \leq n$ , then  $i$  is assigned the values  $m$ ,  $m + 1$ , ...,  $n$  in order, and the body is executed once for each of these values. If  $m > n$ , then the body is skipped and execution continues with the statement after the For... Next loop.

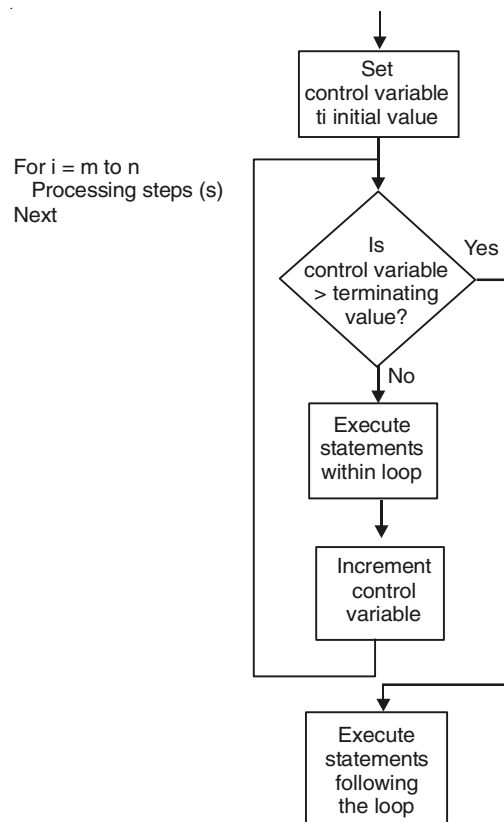
When program execution reaches a For... Next loop, such as the one shown previously, the For statement assigns to the control variable  $i$  the initial value  $m$  and checks to see whether  $i$  is greater than the terminating value  $n$ . If so, then execution jumps to the line following the Next statement. If  $i \leq n$ , the statements inside the loop are executed. Then, the Next statement increases the value of  $i$  by 1 and checks this new value to see if it exceeds  $n$ . If not, the entire process is repeated until the value of  $i$  exceeds  $n$ . When this happens, the program moves to the line following the loop.



Did u know?

The loop does not stop until counter has passed end. If counter is equal to end, the loop continues. The comparison that determines whether to run the block is  $\text{counter} \leq \text{end}$  if step is positive and  $\text{counter} \geq \text{end}$  if step is negative.



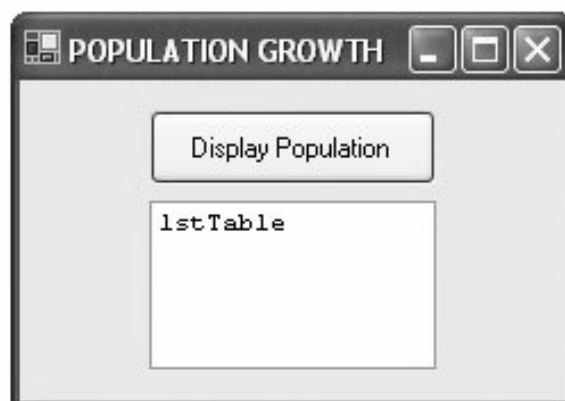


**Figure 6.4:** Pseudocode and Flowchart of a For Next Loop.

The control variable can be any numeric variable. The most common single-letter names are i, j, and k; however, if appropriate, the name should suggest the purpose of the control variable.



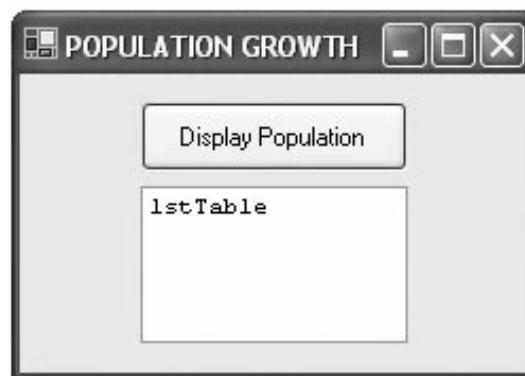
*Example 1:* Suppose the population of a city is 300,000 in the year 2006 and is growing at the rate of three percent per year. The following program displays a table showing the population each year until 2010.



| Object              | Property | Setting            |
|---------------------|----------|--------------------|
| frmPopulation       | Text     | POPULATION GROWTH  |
| btnDisplay lstTable | Text     | Display Population |

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 'Display population from 2006 to 2010
 Dim pop As Double = 300000, yr As Integer
 Dim fmtStr As String = "{0,4}{1,12:N0}"
 For yr = 2006 To 2010
 lstTable.Items.Add(String.Format(fmtStr, yr, pop))
 pop += 0.03 * pop
 Next
End Sub
```

[Run, and click the button.]



The initial and terminating values can be literals, variables, or expressions. For instance, the For statement in the preceding program can be replaced by

```
Dim firstYr As Integer = 2006
Dim lastYr As Integer = 2010
For yr = firstYr To lastYr
```

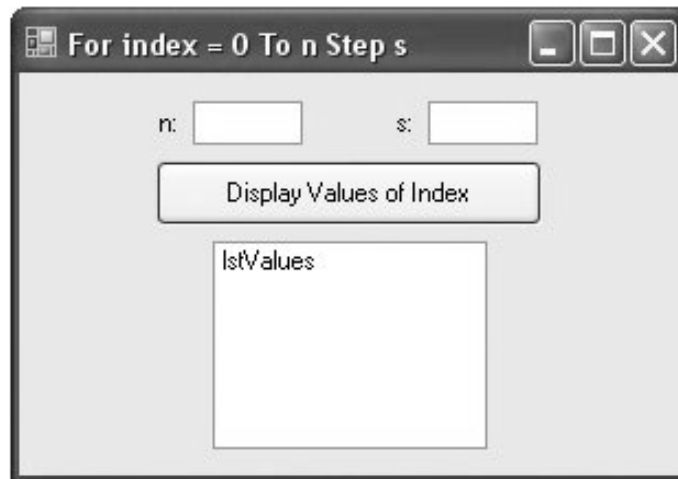
In Example 15, the control variable was increased by 1 after each pass through the loop. A variation of the For statement allows any number to be used as the increment. The statement

```
For i = m To n Step s
```

instructs the Next statement to add s to the control variable instead of 1. The numbers m, n, and s do not have to be whole numbers. The number s is called the step value of the loop. Note: If the control variable will assume values that are not whole numbers, then the variable must be of type Double.



*Example 2:* The following program displays the values of the index of a For... Next loop for terminating and step values input by the user:



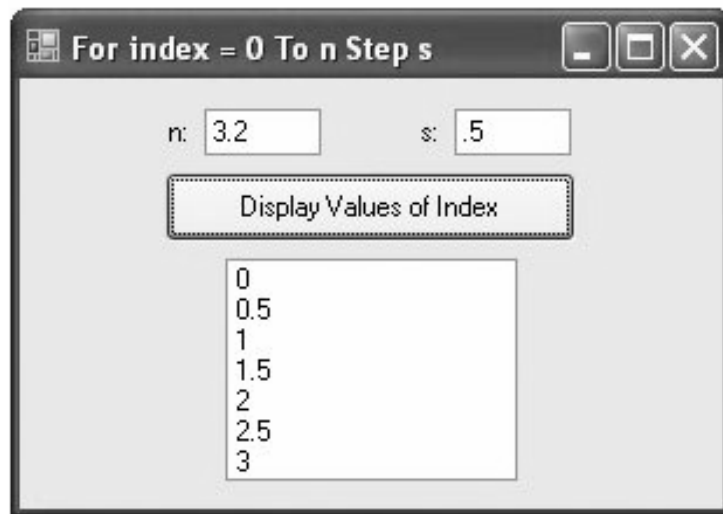
| Object       | Property | Setting                   |
|--------------|----------|---------------------------|
| frmIndex     | Text     | For index = 0 To n Step s |
| lblN txtEnd  | Text     | n:                        |
| lblS txtStep | Text     | s:                        |
| btnDisplay   | Text     | Display Values of Index   |
| lstValues    |          |                           |

```

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 'Display values of index ranging from 0 to n Step s
 Dim n, s, As Double
 Dim index As Double
 n = Cdbl(txtEnd.Text)
 s = Cdbl(txtStep.Text)
 lstValues.Items.Clear()
 For index = 0 To n Step s
 lstValues.Items.Add(index)
 Next
End Sub

```

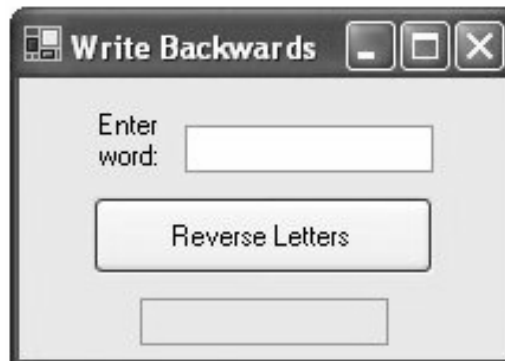
[Run, type 3.2 and .5 into the text boxes, and click the button.]



In the examples considered so far, the control variable was successively increased until it reached the terminating value. However, if a negative step value is used and the initial value is greater than the terminating value, then the control value is decreased until reaching the terminating value. In other words, the loop counts backward or downward.



*Example 3:* The following program accepts a word as input and displays it backwards:



| Object             | Property      | Setting           |
|--------------------|---------------|-------------------|
| FrmBackwards       | Text          | Write Backwards   |
| LblWord            | AutoSize Text | False Enter word: |
| txtWord btnReverse | Text          | Reverse Letters   |
| txtBackwards       | ReadOnly      | True              |

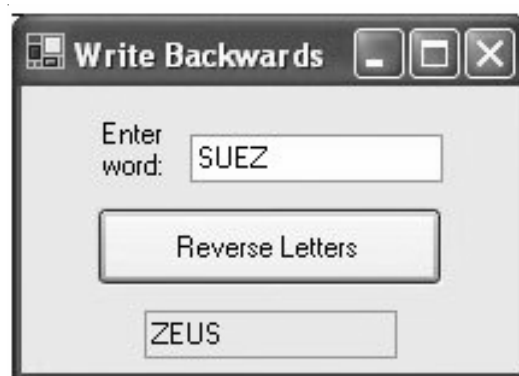
```
Private Sub btnReverse_Click(...) Handles btnReverse.Click
 txtBackwards.Text = Reverse(txtWord.Text)
End Sub
```

```

Function Reverse(ByVal info As String) As String
 Dim m, j As Integer, temp As String = ""
 m = info.Length
 For j = m - 1 To 0 Step -1
 temp &= info.Substring(j, 1)
 Next
 Return temp
End Function

```

[Run, type "SUEZ" into the text box, and click the button.]



The initial and terminating values of a For ...Next loop can be expressions. For instance, the third and fourth lines of the function in Example 17 can be consolidated to

```
For j = info.Length - 1 To 0 Step -1
```

### 6.5.1 Declaration of Control Variables

The control variable of a For ... Next loop can be declared directly in the For statement. A statement of the form

```
For i As DataType = m to n
```

(possibly with a Step clause) both declares the control variable *i* and specifies its initial and terminating values. In this case, however, the scope of the control variable *i* is limited to the body of the For ... Next loop. For instance, in Example 15, the two statements

```
Dim yr As Integer
```

```
For yr = 2006 to 2010
```

can be replaced by the single statement

```
For yr As Integer = 2006 to 2010
```

In Example 16, the two statements

```
Dim index as Double
```

```
For index = 0 to n Step s
```

can be replaced by the single statement

```
For index As Double = 0 to n Step s
```

This feature is new to Visual Basic 2005 and is the preferred way to declare control variables of For... Next loops.



Did u know?

A variable (or simple variable) is a name to which Visual Basic can assign a single value. An array variable is a collection of simple variables of the same type to which Visual Basic can efficiently assign a list of values.

### 6.5.2 Nested For... Next Loops

The body of a For... Next loop can contain any sequence of Visual Basic statements. In particular, it can contain another For... Next loop. However, the second loop must be completely contained inside the first loop and must have a different control variable. Such a configuration is called nested For...Next loops. Figure 4.5 shows several examples of valid nested loops.

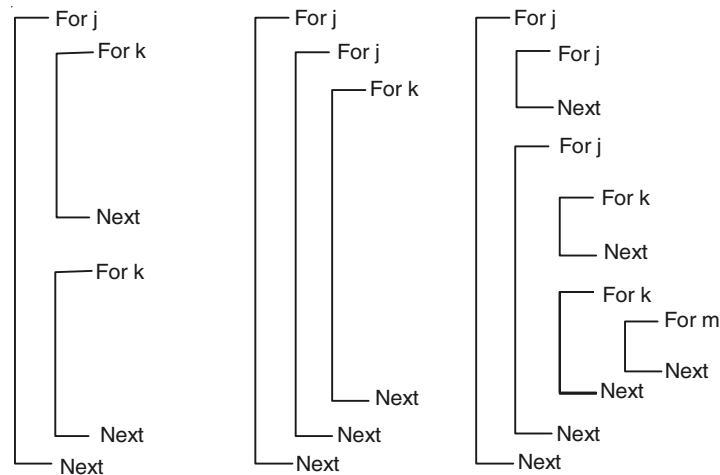


Figure 6.5: Nested For Next Loops.



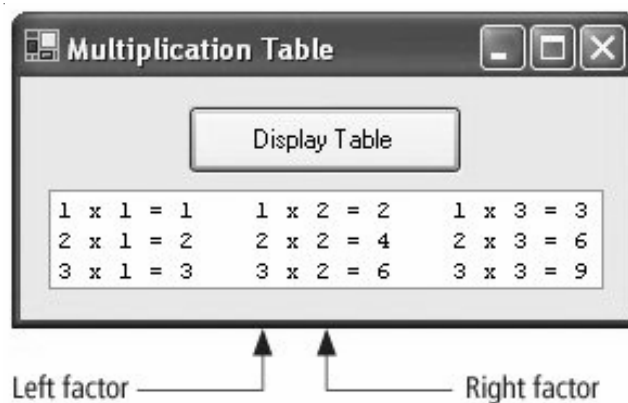
*Example 4:* The following program displays a multiplication table for the integers from 1 to 3. Here *j* denotes the left factors of the products, and *k* denotes the right factors. Each factor takes on a value from 1 to 3. The values are assigned to *j* in the outer loop and to *k* in the inner loop. Initially, *j* is assigned the value 1, and then the inner loop is traversed three times to produce the first row of products. At the end of these three passes, the value of *j* will still be 1, and the value of *k* will have been incremented to 4. The first execution of the outer loop is then complete. Following this, the statement *Next* increments the value of *j* to 2. The statement beginning "For *k*" is then executed. It resets the value of *k* to 1. The second row of products is displayed during the next three executions of the inner loop and so on.



| Object     | Property | Setting              |
|------------|----------|----------------------|
| FrmTable   | Text     | Multiplication Table |
| BtnDisplay | Text     | Display Table        |
| LstTable   | Font     | Courier New          |

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 Dim row, entry As String
 lstTable.Items.Clear()
 For j As Integer = 1 To 3
 row = ""
 For k As Integer = 1 To 3
 entry = j & " x " & k & " = " & (j * k)
 row &= entry & " "
 Next
 lstTable.Items.Add(row)
 Next
End Sub
```

[Run, and press the button.]



## Comments

1. For and Next statements must be paired. If one is missing, the automatic syntax checker will complain with a wavy underline and a message such as "A 'For' must be paired with a 'Next'."
2. Consider a loop beginning with `For i = m To n Step s`. The loop will be executed exactly once if `m` equals `n` no matter what value `s` has. The loop will not be executed at all if `m` is greater than `n` and `s` is positive, or if `m` is less than `n` and `s` is negative.
3. The value of the control variable should not be altered within the body of the loop; doing so might cause the loop to repeat indefinitely or have an unpredictable number of repetitions.
4. Non integer step values can lead to round off errors with the result that the loop is not executed the intended number of times. For instance, a loop beginning with `For i As Double = 1 To 2 Step .1` will be executed only 10 times instead of the intended 11 times. It should be replaced with `For i As Double = 1 To 2.01 Step .1`.
5. Visual Basic provides a way to skip an iteration in a `For ... Next` loop. When the statement `Continue For` is encountered in the body of the loop, execution immediately jumps to the `Next` statement. An analogous statement `Continue Do` is available for `Do` loops. This feature is new in Visual Basic 2005.
6. Visual Basic provides a way to back out of a `For ... Next` loop. When the statement `Exit For` is encountered in the body of the loop, execution jumps immediately to the statement following the `Next` statement. An analogous statement `Exit Do` is available for `Do` loops.

## 6.6 Select Case Blocks

A Select Case block is an efficient decision-making structure that simplifies choosing among several actions. It avoids complex If constructs. If blocks make decisions based on the truth value of a condition; Select Case choices are determined by the value of an expression called a selector. Each of the possible actions is preceded by a clause of the form.

### case valueList

where valueList itemizes the values of the selector for which the action should be taken.



*Did u know?*

We can use multiple expressions or ranges in each Case clause.



*Example 1:* The following program converts the finishing position in a horse race into a descriptive phrase. After the variable `position` is assigned a value from `txtPosition`, Visual Basic searches for the first Case clause whose value list contains that value and executes the succeeding statement. If the value of `position` is greater than 5, then the statement following `Case Else` is executed.





| Object      | Property | Setting                           |
|-------------|----------|-----------------------------------|
| frmRace     | Text     | Horse Race                        |
| lblPosition | AutoSize | False                             |
|             | Text     | Finishing position (1, 2, 3,...): |
| txtPosition |          |                                   |
| btnEvaluate | Text     | Evaluate Position                 |
| txtOutcome  | ReadOnly | True                              |

```


Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click
 Dim position As Integer 'selector
 position = CInt(txtPosition.Text)
 Select Case position
 Case 1
 txtOutcome.Text = "Win"
 Case 2
 txtOutcome.Text = "Place"
 Case 3
 txtOutcome.Text = "Show"

 Case 4, 5
 txtOutcome.Text = "You almost placed in the money."
 Case Else
 Case Else
 txtOutcome.Text = "Out of the money."
 End Select
End Sub

```

[Run, type 2 into the text box, and press the button.]



 *Example 2:* In the following variation of Example, the value lists specify ranges of values. The first value list provides another way to stipulate the numbers 1, 2, and 3. The second value list covers all numbers from 4 on.

```
Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click
 'Describe finishing positions in a horse race
 Dim position As Integer
 position = CInt(txtPosition.Text)
 Select Case position
 Case 1 To 3
 txtOutcome.Text = "In the money. Congratulations."
 Case Is >= 4
 txtOutcome.Text = "Not in the money."
 End Select
End Sub
```

[Run, type 2 into the text box, and press the button.]



A typical form of the Select Case block is

```
Select Case selector
 Case valueList1
```

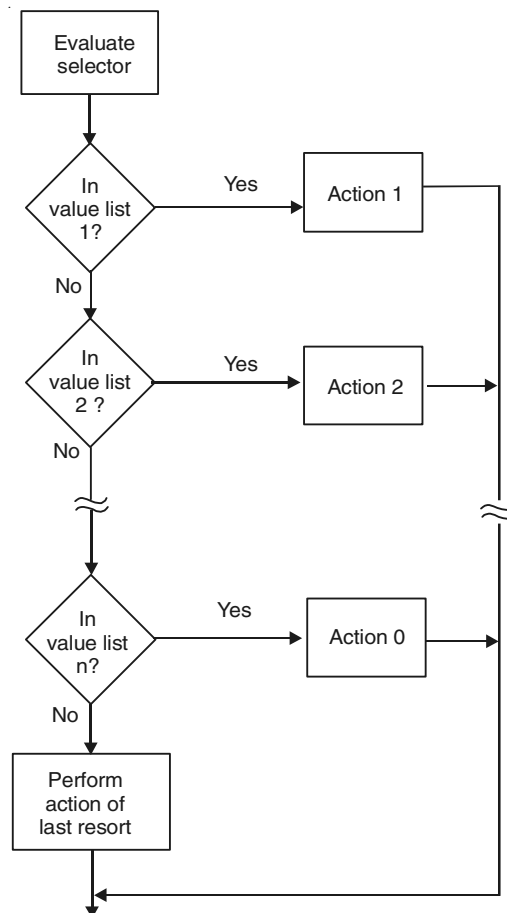
```

 action1
Case valueList2
 action2
Case Else
 action of last resort
End Select

```

where Case Else (and its action) is optional, and each value list contains one or more of the following types of items:

1. a literal;
2. a variable;
3. an expression;
4. an inequality sign preceded by Is and followed by a literal, variable, or expression;
5. a range expressed in the form a To b, where a and b are literals, variables, or expressions.

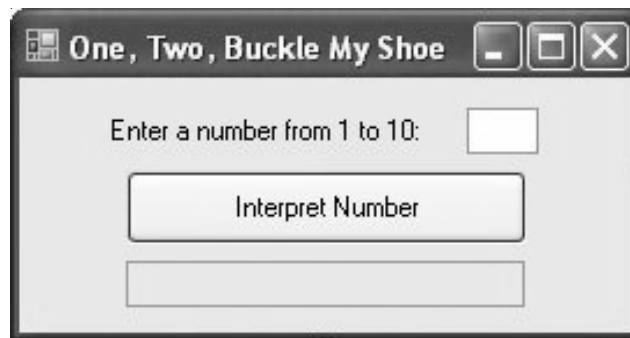


**Figure 6.6:** Flowchart for a Select Case Block.

Different items appearing in the same list must be separated by commas. Each action consists of one or more statements. After the selector is evaluated, Visual Basic looks for the first value-list item including the value of the selector and carries out its associated action. (If the value of the selector appears in two different value lists, only the action associated with the first value list will be carried out.) If the value of the selector does not appear in any of the value lists and there is no Case Else clause, execution of the program will continue with the statement following the Select Case block.



*Example 3:* The following program uses several different types of value lists. With the response shown, the second action was Selected.



| Object       | Property | Setting                      |
|--------------|----------|------------------------------|
| frmRhyme     | Text     | One, Two, Buckle My Shoe     |
| lblEnterNum  | Text     | Enter a number from 1 to 10: |
| txtNumber    |          |                              |
| btnInterpret | Text     | Interpret Number             |
| txtPhrase    | ReadOnly | True                         |

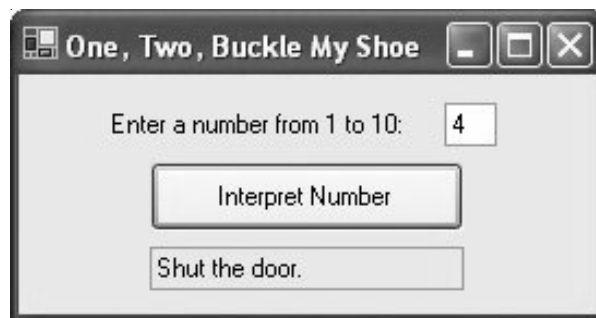
```
Private Sub btnInterpret_Click(...) Handles btnInterpret.Click
 'One, Two, Buckle My Shoe
 Dim x As Integer = 2, y As Integer = 3
 Dim num As Integer
 num = CInt(txtNumber.Text)
 Select Case num
 Case y - x, x
 txtPhrase.Text = "Buckle my shoe."
 Case Is <= 4
 txtPhrase.Text = "Shut the door."
 Case x + y To x * y
 txtPhrase.Text = "Pick up sticks."
```

```

Case 7, 8
 txtPhrase.Text = "Lay them straight."
Case Else
 txtPhrase.Text = "Start all over again."
End Select
End Sub

```

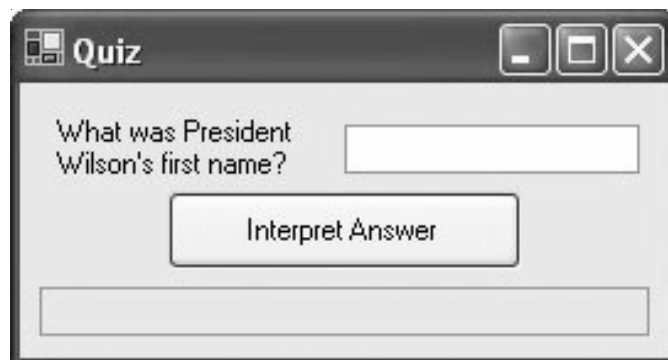
[Run, type 4 into the text box, and press the button.]



In each of the three preceding examples, the selector was a numeric variable; however, the selector also can be a string variable or an expression.



*Example 4:* The following program has the string variable `firstName` as a selector.

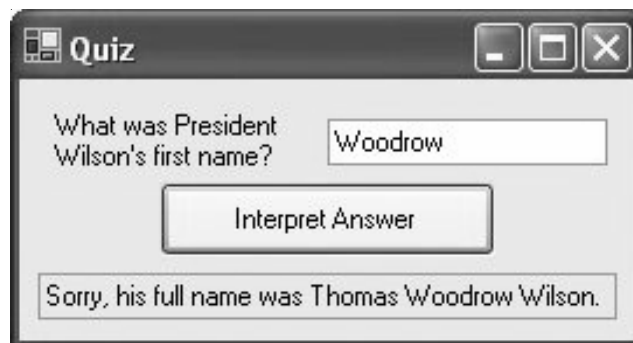


| Object       | Property | Setting                                 |
|--------------|----------|-----------------------------------------|
| frmQuiz      | Text     | Quiz                                    |
| lblQuestion  | AutoSize | False                                   |
|              | Text     | What was President Wilson's first name? |
| txtName      |          |                                         |
| btnInterpret | Text     | Interpret Answer                        |
| txtReply     | ReadOnly | True                                    |

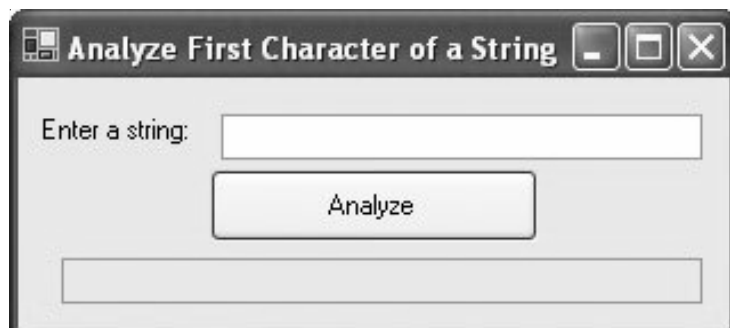
```
Private Sub btnInterpret_Click(...) Handles btnInterpret.Click
 'Quiz
 Dim firstName As String
 firstName = txtName.Text.ToUpper
 Select Case firstName
 Case "THOMAS"
 txtReply.Text = "Correct."
 Case "WOODROW"
 txtReply.Text = "Sorry, his full name was " & _
 "Thomas Woodrow Wilson."

 Case "PRESIDENT"
 txtReply.Text = "Are you for real?"
 Case Else
 txtReply.Text = "Nice try, but no cigar."
 End Select
End Sub
```

[Run, type "Woodrow" into the text box, and press the button.]



*Example 5:* The following program has the string selector `anyString.Substring (0, 1)`. In the sample run, only the first action was carried out, even though the value of the selector was in both of the first two value lists. Visual Basic stops looking as soon as it finds the value of the selector.



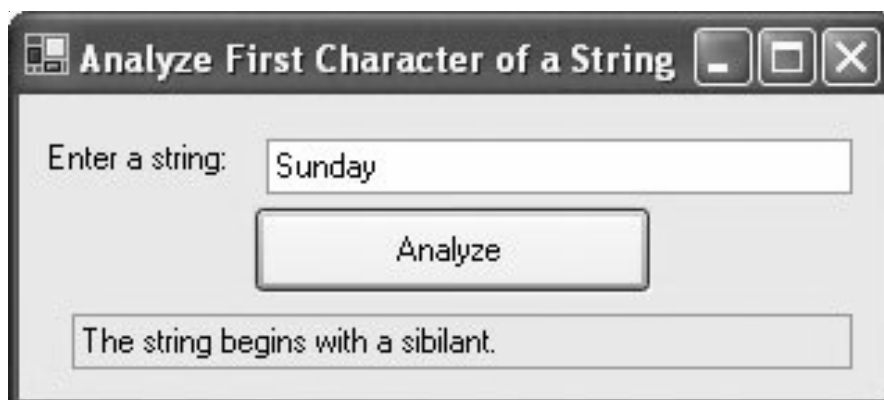
| Object     | Property | Setting                             |
|------------|----------|-------------------------------------|
| frmAnalyze | Text     | Analyze First Character of a String |
| lblEnter   | Text     | Enter a string:                     |
| txtString  |          |                                     |
| btnAnalyze | Text     | Analyze                             |
| txtResult  | ReadOnly | True                                |

```

Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
 'Analyze the first character of a string
 Dim anyString As String
 anyString = txtString.Text.ToUpper
 Select case anyString.Substring(0, 1)
 Case "S", "Z"
 txtResult.Text = "The string begins with a sibilant."
 Case "A" To "Z"
 txtResult.Text = "The string begins with a nonsibilant."
 Case "0" To "9"
 txtResult.Text = "The string begins with a digit."
 Case Is < "0"
 txtResult.Text = "The string begins with a character of " & _
 "ANSI value less than 48."
 Case Else
 txtResult.Text = "The string begins with : ; < = > " & _
 " ? @ [\] ^ _ or ' . "
 End Select
End Sub

```

[Run, type "Sunday" into the text box, and press the button.]





*Example 6:* The color of the beacon light atop Boston's John Hancock Building forecasts the weather according to the following rhyme:

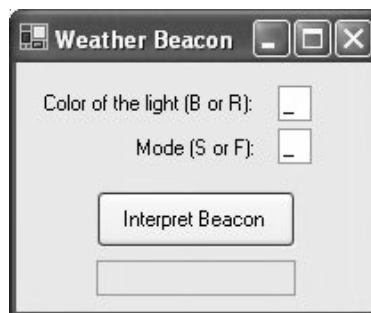
Steady blue, clear view.

Flashing blue, clouds due.

Steady red, rain ahead.

Flashing red, snow instead.

The following program requests a color (Blue or Red) and a mode (Steady or Flashing) as input and displays the weather forecast. The program contains a Select Case block with a string expression as selector.



| Object       | Property | Setting                      |
|--------------|----------|------------------------------|
| frmWeather   | Text     | Weather Beacon               |
| lblColor     | Text     | Color of the light (B or R): |
| mtxtColor    | Mask     | L                            |
| lblMode      | Text     | Mode (S or F ):              |
| mtxtMode     | Mask     | L                            |
| btnInterpret | Text     | Interpret Beacon             |
| txtForecast  | ReadOnly | True                         |

```
Private Sub btnInterpret_Click(...) Handles btnInterpret.Click
 'Interpret a weather beacon
 Dim color, mode As String
 color = mtxtColor.Text
 mode = mtxtMode.Text
 Select Case mode.ToUpper & color.ToUpper
 Case "SB"
 txtForecast.Text = "CLEAR VIEW"
 Case "FB"
 txtForecast.Text = "CLOUDS DUE"
 Case "SR"
```



```

 txtForecast.Text = "RAIN AHEAD"
 Case "FR"
 txtForecast.Text = "SNOW AHEAD"
 End Select
End Sub

```

[Run, type "R" and "S" into the masked text boxes, and press the button.]



*Example 7:* Select case is useful in defining functions that are not determined by a formula. The following program assumes that the current year is not a leap year:



| Object     | Property | Setting        |
|------------|----------|----------------|
| frmSeasons | Text     | Seasons        |
| lblSeason  | Text     | Season:        |
| TxtSeason  |          |                |
| btnNumber  | Text     | Number of Days |
| txtNumDys  | ReadOnly | True           |

```
Private Sub btnNumber_Click(...) Handles btnNumber.Click
 'Determine the number of days in a season
 Dim season As String
 season = txtSeason.Text
 txtNumDays.Text = season & " has " & NumDays(season) & " days."
End Sub

Function NumDays(ByVal season As String) As Integer
 'Look up the number of days in a given season
 Select case season.ToUpper
 Case "WINTER"
 Return 87
 Case "SPRING"
 Return 92
 Case "SUMMER", "AUTUMN", "FALL"
 Return 93
 End Select
End Function
```

[Run, type "Summer" into the text box, and press the button.]



### Comments

1. In a Case clause of the form Case b To c, the value of b should be less than or equal to the value of c. Otherwise, the clause is meaningless.
2. If the word Is, which should precede an inequality sign in a value list, is accidentally omitted, the editor will automatically insert it when checking the line.
3. The items in the value list must evaluate to a literal of the same type as the selector. For instance, if the selector evaluated to a string value, as in
4. Dim firstName As String
5. firstName = txtBox.Text
6. Select Case firstName  
then the clause Case firstName.Length would be meaningless.

7. Variables are rarely declared inside an If ... Then block or a Select Case block. If so, such a variable has block-level scope; that is, the variable cannot be referred to by code outside of the block.



### Case Study

#### Wroba Case Study

The WROBA (WRox Online Banking Application) case study used in this book uses a theme you may be familiar with—Online banking. In fact, what we are going to develop during the first part of the book is a desktop based application—it still counts as online, in that it will enable a user sitting at their desk to carry out various banking activities, even though it will have a Visual Basic form type user interface.

The functionality of the case study was arrived at by careful analysis of the business requirements of a hypothetical client (Big Bank) and this is covered in Chapter 6. But so we're all clear about where we are heading, let's highlight some of the major parts of WROBA. There are two sides to the application;

a customer side and an administration side.

The customer side will:

- Show the status of the users accounts (two accounts are permissible Checking and Saving, but only one of each per card)
- Allow previous transactions to be viewed
- Allow user passwords to be changed
- Allow users to add or edit payee details
- Allow users to move money between accounts
- Allow users to pay bills to payees

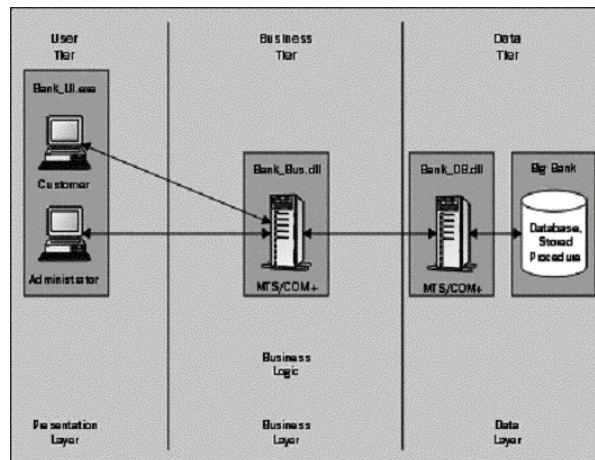
The administration side of the application offers a separate login for designated administrators, and allows the administrator to:

- Add and delete new accounts (as represented by cards)
- Lock and unlock cards that have fallen foul either of the login system (too many wrong login attempts) or the banking authorities
- Add and edit back details (this feature is not fully implemented in our fledgling application) Let's have a look at how this initial version relates to the 3-tier architecture we've previously looked at:

The data tier consists of two units—a SQL Server 7.0 database (called Big Bank) that contains both data and stored procedures for accessing and operating on that data, and the Bank\_DB.Dll component.

The business tier consists of one COM component—Bank\_Bus.dll.

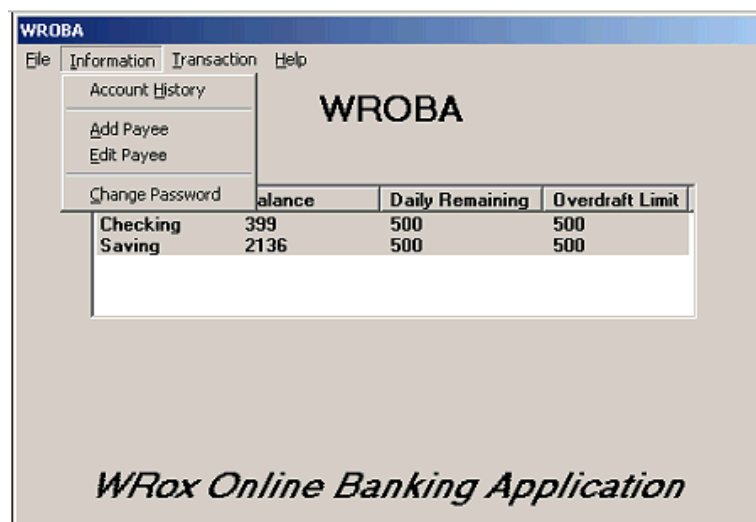
The user tier functionality is provided by Bank\_UI.exe—although it could be split into two projects, we have dealt with it as one.



In terms of the Windows DNA model, we're using SQL Server 7.0 to manage the data, UDA as we code the Bank\_DB.dll component, and MTS (or COM+ depending on the platform) to provide an environment for Bank\_Bus.dll and Bank\_DB.dll to operate in. We haven't talked about ASP and IIS here, but when we show how to extend the WROBA case study and Web-enable, those products will also make an appearance.

Since it's a mere case study, there is some simplification. There are areas, which we highlight, that the enthusiastic reader may like to improve upon. We've made no attempts to implement deposit and withdrawal functionality, since there is no cash drawer, and no electronic checks or credit card accounts are implemented.

Just as a taste of what we'll be building, the following illustration shows the Information menu of the customer interface part of the application:



### Questions

1. Explain the WROBA.
2. What do you understand to online banking system?

## 6.7 Summary

- Loop is one of the most important structure in visual basic, as it repeats a sequence of statement either as long as until a certain condition is true.
- Do-while loops repeat the sequence of statement as long as or until certain condition is true.
- Next loops are easy to read and write and also have features that make them ideal for certain common tasks.

## 6.8 Keywords

**Do loops:** It repeats a sequence of statements either as long as or until a certain condition is true

**Else If clauses:** An extension of the If block allows for more than two possible alternatives with the inclusion of Elseif clause

**If block:** If block allows a program to decide on course of action based on a course of action based on whether certain conditions are true or false.

**Loop:** A loop, one of the most important structures in Visual Basic, is used to repeat a sequence of statements a number of times.

**User-Defined Type (UDT):** In previous versions of Visual Basic a Structure was called a user-defined type (UDT).

**While Wend statement:** The While Wend statement repeats a block of code while a condition is true. The condition is always evaluated before entering the loop. If it is not fulfilled, the loop is never executed.



Lab Exercise

Write a program using if-else statement.

## 6.9 Self Assessment Questions

- In general, a Structure type is defined by:
  - Structure block of the form
  - Structure block of pixel
  - Structure block of the image
  - None of the above.
- Visual Basic a Structure was called as a user-defined type (UDT).
  - True
  - False
- Subvariables of the Structure type is called a.....
  - sub member
  - group
  - member
  - none
- An ..... allows a program to decide on a course of action based on whether certain conditions are true or false.
  - If-else
  - If block
  - For—next loop
  - Do while loops

5. The ..... repeats a sequence of statements either as long as or until a certain condition is true.
  - (a) Do loop
  - (b) Do while loops
  - (c) If block
  - (d) If-else
6. For... Next loops are easy to read and write, and have features that make them ideal for certain common tasks.
  - (a) For...Next loops
  - (b) If-else
  - (c) Do While loops
  - (d) While-While

### **6.10 Review Questions**

1. With a single loop, determine whether the array is in ascending order, descending order, both, or neither.
2. What is an array? Explain declaration and processing of one and two dimensional array. How to create dynamic array?
3. Explain If-else block with the help of suitable examples.
4. Variables are rarely declared inside an If ... Then block or a Select Case block. If so, such a variable has block-level scope; that is, the variable cannot be referred to by code outside of the block.
5. What do you mean by Do while loops and give its syntax also?

### **Answers for Self Assessment Questions**

- |        |        |        |        |
|--------|--------|--------|--------|
| 1. (a) | 2. (a) | 3. (c) | 4. (b) |
| 5. (a) | 6. (a) |        |        |

### **6.11 Further Readings**



Books

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net."



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

## Unit 7: Arrays in Visual Basic

### CONTENTS

Objectives

Introduction

7.1 Declaring Arrays

7.1.1 Fixed-sized Arrays

7.1.2 Multidimensional Arrays

7.1.3 Static and Dynamic Arrays

7.1.4 Arrays within UDTs

7.2 ReDim and Preserve Function

7.2.1 Using an Array as a Frequency

7.3 Control Array

7.3.1 Iterating on the Items of a Control Array

7.4 Summary

7.5 Keywords

7.6 Self Assessment Questions

7.7 Review Questions

7.8 Further Readings

### Objectives

*After studying this unit, you will be able to:*

- Discuss declaring arrays
- Explain ReDim and Preserve function
- Understand the control array in visual basic

### Introduction

By definition, an array is a list of variables, all with the same data type and name. When we work with a single item, we only need to use one variable. However, if we have a list of items which are of similar type to deal with, we 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, we might have difficulty in declaring 100 different names; this is a waste of time and efforts. So, instead of declaring one hundred different variables, we need to declare only one array. We differentiate each item in the array by using subscript, the index value of each item, for example name(1), name(2), name(3)... etc., which will make declaring variables streamline and much systematic.

## **7.1 Declaring Arrays**

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

There are two types of arrays in Visual Basic namely:

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

### **7.1.1 Fixed-sized Arrays**

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

Declaring a fixed-array

Dim numbers (5) As Integer

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

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

Dim numbers (1 To 6 ) As Integer

In the above statement, an array of 10 elements is declared but with indexes running from 1 to 6. A public array can be declared using the keyword Public instead of Dim as shown below.

Public numbers (5) As Integer

### **7.1.2 Multidimensional Arrays**

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

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

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

Dim AvgMarks (50, 50)

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

Dim Marks (101 To 200, 1 To 100)

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

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



### 7.1.3 Static and Dynamic Arrays

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

```
' This is a static array.
```

```
Dim Names(100) As String
```

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

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

```
' An array defined in a BAS module (with Private scope)
```

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

```
...
```

```
Sub Main()
```

```
' Here you create the array.
```

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

```
End Sub
```

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

```
Sub PrintReport()
```

```
This array is visible only to the procedure.
```

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

```
' ...
```

```
End Sub
```

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

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

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

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

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

```
ReDim Cells(1 To 100, 10) As Integer
```

```
...
```

```
ReDim Preserve Cells(1 To 100, 20) As Integer ' This works.
```

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

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

if the array is static, its elements are set to 0 or to empty strings.

You can use the LBound and UBound functions to retrieve the lower and upper indices. If the array has two or more dimensions, you need to pass a second argument to these functions to specify the dimension you need:

```
Print LBound(Cells, 1) ' Displays 1, lower index of 1st dimension
```

```
Print LBound(Cells) ' Same as above
```

```
Print UBound(Cells, 2) ' Displays 20, upper index of 2nd dimension
```

```
' Evaluate total number of elements.
```

```
NumEls = (UBound(Cells) _ LBound(Cells) + 1) * _
```

```
(UBound(Cells, 2) _ LBound(Cells, 2) + 1)
```

### 7.1.4 Arrays within UDTs

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

```
Type MyUDT
```

```
StaticArr(100) As Long
```

```
DynamicArr() As Long
```

```
End Type
```

```
...
```

```
Dim udt As MyUDT
```

```
' You must DIMension the dynamic array before using it.
```

```
ReDim udt.DynamicArr(100) As Long
```

```
' You don't have to do that with static arrays.
```

```
udt.StaticArr(1) = 1234
```

The memory needed by a static array is allocated within the UDT structure; for example, the StaticArr array in the preceding code snippet takes exactly 400 bytes. Conversely, a dynamic array in a UDT takes only 4 bytes, which form a pointer to the memory area where the actual data is stored. Dynamic arrays are advantageous when each individual UDT variable might host a different number of array items. As with all dynamic arrays, if you don't dimension a dynamic array within a UDT before accessing its items, you get an error 9—"Subscript out of range".

## 7.2 ReDim and Preserve Function

After an array has been declared, its size (but not its type) can be changed with a statement of the form

```
ReDim arrayName(m)
```

where arrayName is the name of the already declared array and m is an Integer literal, variable, or expression. Note: Since the type cannot be changed, there is no need for an "As dataType" clause at the end of the ReDim statement.

Visual Basic allows you to declare an array without specifying an upper bound with a statement of the form

```
Dim arrayName() As varType
```

Later, the size of the array can be stipulated with a ReDim statement.

The ReDim statement has one shortcoming: It causes the array to lose its current contents. That is, it resets all String values to Nothing and resets all numeric values to 0. This situation can be remedied by following ReDim with the word Preserve. The general form of a ReDim Preserve statement is

```
ReDim Preserve arrayName(m)
```

Of course, if you make an array smaller than it was, data in the eliminated elements will be lost.



*Did u know?*

We can use the ReDim statement repeatedly to change the number of elements and dimensions in an array. However, we can't declare an array of one data type and later use ReDim to change the array to another data type, unless the array is contained in a Variant.



**Example 1:** The following program reads the names of the winners of Super Bowl games from a file and places them into an array. The user can type a team's name into a text box and then display the numbers of the Super Bowl games won by that team. The user has the option of adding winners of subsequent games to the array of winners. The program uses the text file SBWINNERS.TXT whose lines contain the names of the winners in order. That is, the first four lines of the file contain the names Packers, Packers, Jets, and Chiefs.

```
Dim teamName() As String
Dim upperBound As Integer
Private Sub frmBowl_Load(...) Handles MyBase.Load
 Dim sr As IO.StreamReader = IO.File.OpenText("SBWINNERS.TXT")
 Dim name As String, numLines As Integer
 'Count number of lines in the file and assign it minus one to upperBound
 numLines = 0
 Do While (sr.Peek <> -1)
```

```
 name = sr.ReadLine
 numLines += 1
 Loop
 upperBound = numLines - 1
 sr.Close()
 'Specify the title bar of the form
 Me.Text = "First " & upperBound + 1 & " Super Bowls"
 'Specify the caption of the Next Winner button
 btnNextWinner.Text = "Add Winner of Game " & upperBound + 2
 'Specify the size of the array and fill it with the entries of the
file
 ReDim teamName(upperBound)
 sr = IO.File.OpenText("SBWINNERS.TXT")
 For i As Integer = 0 To upperBound
 teamName(i) = sr.ReadLine
 Next
 sr.Close()
End Sub

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 'Display the numbers of the games won by the team in the text box
 lstGamesWon.Items.Clear()
 For i As Integer = 0 To upperBound
 If teamName(i) = txtName.Text Then
 lstGamesWon.Items.Add(i + 1)
 End If
 Next
End Sub

Private Sub btnNextWinner_Click(...) Handles btnNextWinner.Click
 'Add winner of next Super Bowl to the array
 Dim prompt As String
 upperBound += 1
 'Add one more element to the array
 ReDim Preserve teamName(upperBound)
 'Request the name of the next winner
 prompt = "Enter winner of game #" & upperBound + 1
 teamName(upperBound) = InputBox(prompt, , "")
 'Update the title bar of the form and the caption of the button
 'Note: "Me" refers to the form.
```

```

Me.Text = "First " & upperBound + 1 & " Super Bowls"
btnNextWinner.Text = "Add Winner of Game " & upperBound + 2
End Sub

```

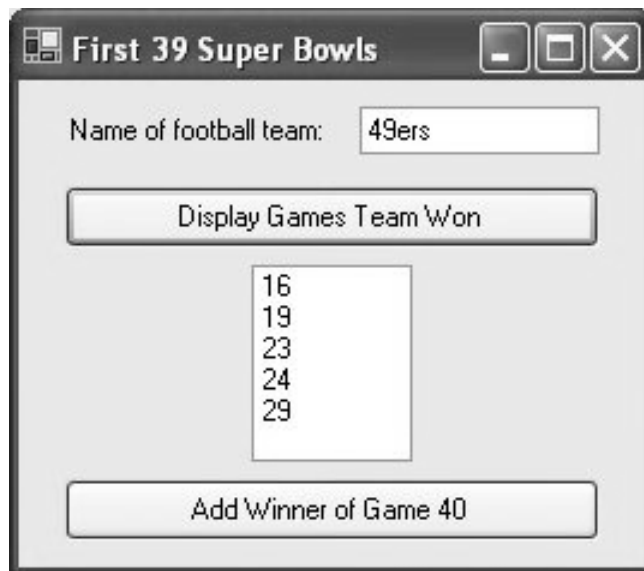
[Run, type "49ers" into the text box, and press the Display button. Then, feel free to add subsequent winners. Your additions will be taken into account when you next press the Display button.]



Caution

The **ReDim** statement can appear only at procedure level. This means you can redefine arrays inside a procedure but not at class or module level.

### 7.2.1 Using an Array as a Frequency



An array can be used as either a checklist or a frequency table, as in the next example.



*Example 2:* The following program requests a sentence as input and records the frequencies of the letters occurring in the sentence. The first element of the array `charCount()` holds the frequency with which the letter A occurs in the sentence, and so on. Recall that the function `Asc` associates each character with its position in the ANSI table.

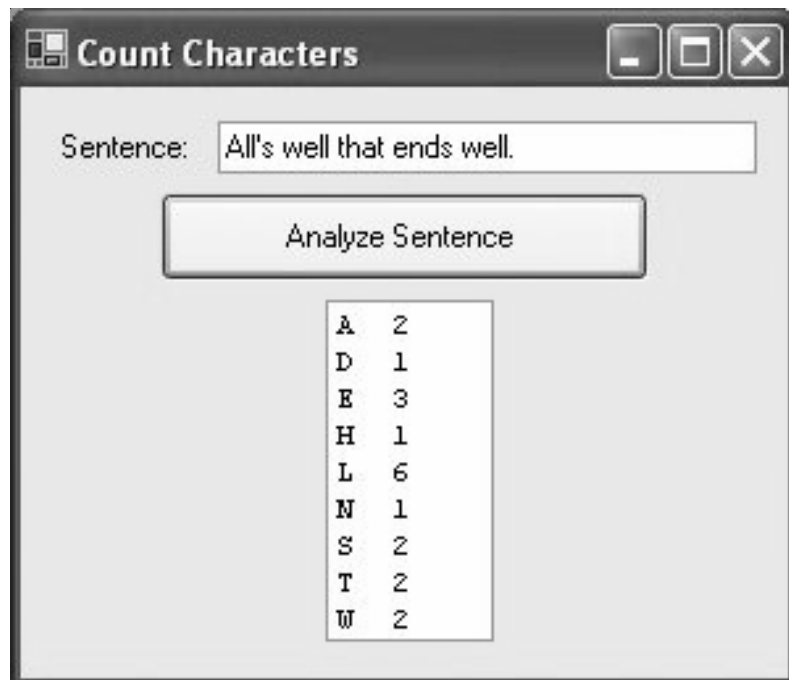
```

Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
 'Count occurrences of the various letters in a sentence
 Dim index As Integer
 Dim sentence, letter As String
 Dim charCount(25) As Integer

```

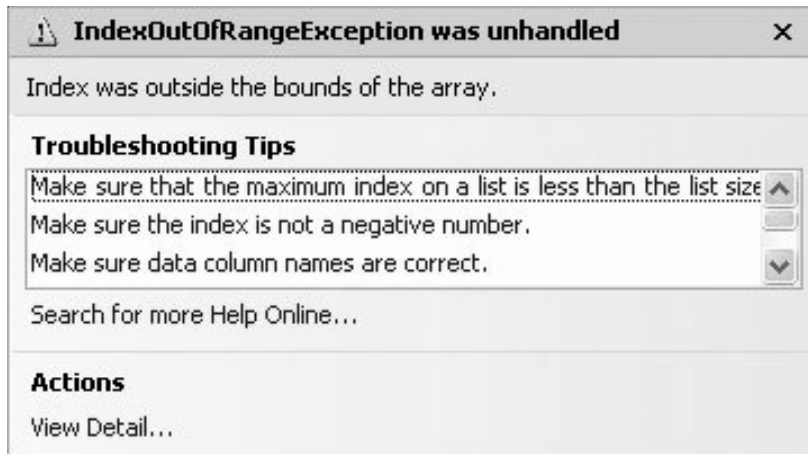
```
'Examine and tally each letter of the sentence
sentence = (txtSentence.Text).ToUpper
For letterNum As Integer = 1 To sentence.Length
 letter = sentence.Substring(letterNum - 1, 1)
 If (letter >= "A") And (letter <= "Z") Then
 index = Asc(letter) - 65 'The ANSI value of "A" is 65
 charCount(index) += 1
 End If
Next
'List the tally for each letter of alphabet
lstCount.Items.Clear()
For i As Integer = 0 To 25
 letter = Chr(i + 65)
 If charCount(i) > 0 Then
 lstCount.Items.Add(letter & " " & charCount(i))
 End If
Next
End Sub
```

[Run, type a sentence into the text box, and press the button.]



## Comments

1. Using a subscript greater than the upper bound of an array is not allowed. For instance, the following lines of code produce the error dialog box shown immediately thereafter:
2. `Dim trees() As String = {"Sequoia", "Redwood", "Spruce"}`
3. `txtBox.Text = trees(5)`



4. If `arrayOne()` and `arrayTwo()` have been declared with the same data type, then the statement  
`arrayOne = arrayTwo`  
 makes `arrayOne()` an exact duplicate of `arrayTwo()`. It will have the same size and contain the same information.
5. An array can require a large block of memory. After the array is no longer needed, the statement `Erase array name` can be executed to release all memory allocated to the array.

## 7.3 Control Array

Control arrays can be created at run time using the statements

- Load object (Index %)
- Unload object (Index %)

Where object is the name of the control to add or delete from the control array. Index % is the value of the index in the array. The control array to be added must be an element of the existing array created at design time with an index value of 0. When a new element of a control array is loaded, most of the property settings are copied from the lowest existing element in the array.

Following example illustrates the use of the control array.

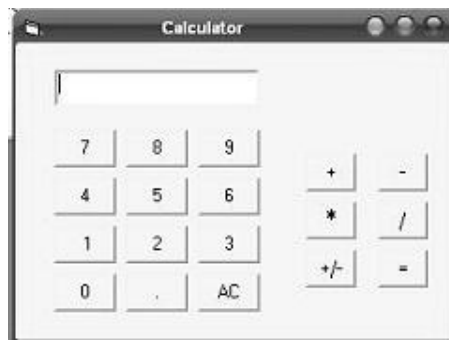
- Open a Standard EXE project and save the Form as `Calculator.frm` and save the Project as `Calculator.vbp`.

- Design the form as shown below:

| Object        | Property                 | Setting                     |
|---------------|--------------------------|-----------------------------|
| Form          | Caption<br>Name          | Calculator<br>frmCalculator |
| CommandButton | Caption<br>Name<br>Index | 1<br>cmd<br>0               |
| CommandButton | Caption<br>Name<br>Index | 2<br>cmd<br>1               |
| CommandButton | Caption<br>Name<br>Index | 3<br>cmd<br>2               |
| CommandButton | Caption<br>Name<br>Index | 4<br>cmd<br>3               |
| CommandButton | Caption<br>Name<br>Index | 5<br>cmd<br>4               |
| CommandButton | Caption<br>Name<br>Index | 6<br>cmd<br>5               |
| CommandButton | Caption<br>Name<br>Index | 7<br>cmd<br>6               |
| CommandButton | Caption<br>Name<br>Index | 8<br>cmd<br>7               |
| CommandButton | Caption<br>Name<br>Index | 9<br>cmd<br>8               |
| CommandButton | Caption<br>Name<br>Index | 0<br>cmd<br>10              |
| CommandButton | Caption<br>Name<br>Index | .<br>cmd<br>11              |



|               |                 |                         |
|---------------|-----------------|-------------------------|
| CommandButton | Caption<br>Name | AC<br>cmdAC             |
| CommandButton | Caption<br>Name | +<br>cmdPlus            |
| CommandButton | Caption<br>Name | -<br>cmdMinus           |
| CommandButton | Caption<br>Name | *<br>cmdMultiply        |
| CommandButton | Caption<br>Name | /<br>cmdDivide          |
| CommandButton | Caption<br>Name | +/-<br>cmdNeg           |
| TextBox       | Name<br>Text    | txtDisplay<br>( empty ) |
| CommandButton | Caption<br>Name | =<br>cmdEqual           |



The following variables are declared inside the general declaration

```
Dim Current As Double
Dim Previous As Double
Dim Choice As String
Dim Result As Double
```

The following code is entered in the cmd\_Click( ) (Control Array) event procedure

```
Private Sub cmd_Click(Index As Integer)
txtDisplay.Text = txtDisplay.Text & cmd(Index).Caption
'&is the concatenation operator
Current = Val(txtDisplay.Text)
```

End Sub

The following code is entered in the cmdAC\_Click ( ) event procedure

```
Private Sub cmdAC_Click()
Current = Previous = 0
txtDisplay.Text = ""
End Sub
```

The below code is entered in the cmdNeg\_Click( ) procedure

```
Private Sub cmdNeg_Click()
Current = -Current
txtDisplay.Text = Current
End Sub
```

The following code is entered in the click events of the cmdPlus, cmdMinus, cmdMultiply, cmdDivide controls respectively.

```
Private Sub cmdDivide_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "/"
End Sub

Private Sub cmdMinus_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "-"
End Sub

Private Sub cmdMultiply_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "*"
End Sub

Private Sub cmdPlus_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "+"
End Sub
```

To print the result on the text box, the following code is entered in the cmdEqual\_Click ( ) event procedure.

```
Private Sub cmdEqual_Click()
 Select Case Choice
 Case "+"
 Result = Previous + Current
 txtDisplay.Text = Result
 Case "-"
 Result = Previous - Current
 txtDisplay.Text = Result
 Case "*"
 Result = Previous * Current
 txtDisplay.Text = Result
 Case "/"
 Result = Previous / Current
 txtDisplay.Text = Result
 End Select
 Current = Result
End Sub
```

Save and run the project. On clicking digits of user's choice and an operator button, the output appears.

### 7.3.1 Iterating on the Items of a Control Array

Control arrays often let you save many lines of code because you can execute the same statement, or group of statements, for every control in the array without having to duplicate the code for each distinct control. For example, you can clear the contents of all the items in an array of TextBox controls as follows:

```
For i = txtFields.LBound To txtFields.UBound
 txtFields(i).Text = ""
Next
```

Here you're using the LBound and UBound methods exposed by the control array object, which is an intermediate object used by Visual Basic to gather all the controls in the array. In general, you shouldn't use this approach to iterate over all the items in the array because if the array has holes in the Index sequence an error will be raised. A better way to loop over all the items of a control array is using the For Each statement:

```
Dim txt As TextBox
For Each txt In txtFields
 txt.Text = ""
Next
```

A third method exposed by the control array object, Count, returns the number of elements it contains. It can be useful on several occasions (for example, when removing all the controls that were added dynamically at run time):

```
' This code assumes that txtField(0) is the only control that was
' created at design time (you can't unload it at run time).
```

```
Do While txtFields.Count > 1
Unload txtFields(txtFields.UBound)
Loop
```



A Form object is of type Control; you can also pass a Form as an argument.



#### Case Study

##### Adding the Array synthesis tool in Antenna Magus

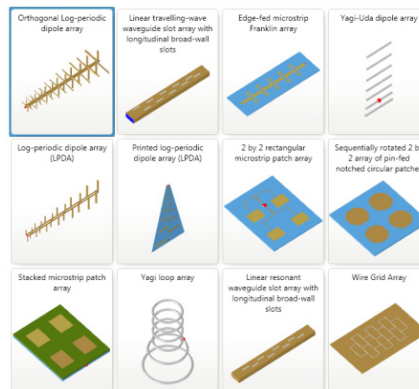
Sometime in early in 2009, we all sat around the table and decided that we would be including an array calculator in Magus, because people kept on asking us: “Can Magus do arrays?” So it seemed like a feature that a lot of people really needed.

But what does that mean? When people ask whether Magus *does* arrays, what exactly is it supposed to *do*?

To find out, we interviewed people—at the coffee machine, over lunch and more formally in the boardroom. We asked questions like: What are common arrays? Why would you want one? How could Magus help you? If we told you that Magus now included an array calculator, what do you think it would do? Where do you start when you design an array?

Everyone had different answers. Nobody could tell us what they wanted specifically, but at least we could form an idea of the kinds of things that engineers thought about when thinking of arrays.

Magus had always included some basic arrays as antennas – like the LPDA, the Yagi, little 2-by-2 patch arrays and later on the slotted waveguide array and our beloved *braairooster* antenna. But of course, these antennas have specific shapes and parameters.



Array antennas in the Magus database

An array calculator, on the other hand, would have to allow you to specify (almost) any number of radiators in different layouts and then tell you what the overall performance is.

With our first design we started off with the following two premises:

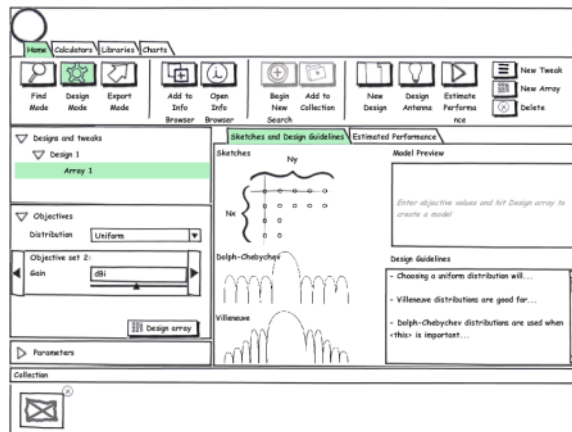
- anything is possible and
- an array is a bunch of antennas

To design an array, therefore, you first need to know what your elemental antenna looks like. After that you can decide how many you want and how they should be laid out.

In many ways, arrays are just like antennas. Hence we thought it would be great to incorporate the array design workflow into the existing antenna design workflow, in other words: make arrays work just like antennas.

The user would design an elemental antenna, like a patch or a dipole. He could then choose to make an array of it, either by specifying the number of elements in the x- and y-directions or allowing Magus to tell him the number of elements needed, their positions and their excitations.

The first prototype (shown in the image below) was drawn up in Balsamiq and put in front of five users. In the next blog I will write on what the usability tests told us but have a look at the prototype so long and write down (or comment) what you thought about it. Click on the image below to enlarge.



Array tool: Prototype

#### Questions

1. How to use Array antennas in the Magus database?
2. Explain the concept of Array tool prototype.

## 7.4 Summary

- Array occupy space in memory.
- Array can have multiple dimensions.

- ReDim is using for declaring or resizing the array size.
- The control array to be added must be an element of the existing array created at design time with an index value of 0.

## 7.5 Keywords

**Dynamic array:** The size of the array can be changed at the run time size changes during the program execution.

**Fixed-size array:** The size of array always remains the same-size doesn't change during the program execution.

**LBound and UBound functions:** The LBound and UBound functions to retrieve the lower and upper indices. If the array has two or more dimensions.

**Multidimensional arrays:** Multidimensional arrays is to represent tables of values consisting of information arranged in rows and columns.

**UDT structures:** UDT structures can include both static and dynamic arrays.



Lab Exercise

1. Write a program using if-else statement.
2. How to define an array? Give example.

## 7.6 Self Assessment Questions

1. Which of the properties in a control's list of properties is used to give the control a meaningful name?  
(a) Text (b) ContextMenu  
(c) ControlName (d) Name
2. Select Case choices are determined by the value of an expression called a disselector.  
(a) True (b) False
3. The size of an array can be changed at the run time—size changes during the program execution.  
(a) True (b) False
4. Tables or arrays that require—index to identify a particular element are called two dimensional array.  
(a) Three (b) One  
(c) Two (d) None of these
5. Multidimensional arrays is to represent tables of values consisting of information arranged in .....

## 7.7 Review Questions

1. Write code to declare an array with upper bound 10 and place the names of the five Great Lakes into the first five elements. Use one or two lines of code.
2. Suppose the Integer array quantities() has been declared with an upper bound of 100 and data have been placed into several elements of the array. Write one line of code that will restore the values of all the elements to their default value of 0.
3. Write code to declare an array of size 20 and place the names Jerry, George, Elaine, and Kramer into the first four elements. Use one or two lines of code.
4. Declare the string array marx() with upper bound 3 so that the array is visible to all parts of the program. Assign the four values Chico, Harpo, Groucho, and Zeppo to the array before any buttons are pressed.
5. Declare the string array stooges() to have size 3 so that the array is local to the event procedure btnStooges\_Click. Assign the three values Moe, Larry, and Curly to the array as soon as the button is clicked.
6. The arrays a() and b() have been declared to have upper bound 4, and values have been assigned to a(0) through a(4). Store these values in array b() in reverse order.
7. Write a program that asks the user for a month by number and then displays the name of that month. For instance, if the user inputs 2, the program should display February.  
**Hint:** Create an array of 12 strings, one for each month of the year.
8. Assuming that the array is in ascending order, determine how many numbers appear more than once in the array?
9. Write a procedure to place all the 40 numbers from arrays a() and b() into c() so that c() is also in ascending order. The array c() could contain duplications.
10. Explain about arrays.
11. If you intend to dynamically create a TextBox control from a control array at runtime, what two things must you do at design time for this to work?

## Answers for Self Assessment Questions

1. (d)
2. (b)
3. (a)
4. (c)
5. Rows and columns.

## 7.8 Further Readings



Books

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net."



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 8: Understanding Function and Procedure**

### **CONTENTS**

Objectives

Introduction

8.1 Meaning of Understanding Function and Procedure

8.2 Function Procedure

8.3 User-Defined Function Having Several Parameters

8.4 User-Defined Function Having No Parameters

8.5 Comparing Function Procedures with Sub Procedures

8.6 Collapsing a Procedures with a Region Directive

8.7 Modules

8.8 Summary

8.9 Keywords

8.10 Self Assessment Questions

8.11 Review Questions

8.12 Further Readings

### **Objectives**

*After studying this unit, you will be able to:*

- Understanding function and procedure definition
- Discuss user defined functions with several parameters
- Explain the function procedures and its features.

### **Introduction**

In your code you may frequently need to use the same instructions, by declaring a function/ procedure and typing the needed instructions into it you can replace the repeated instructions with a call to this function. Functions/Procedures are a very useful part of Visual Basic programming because your code will be smaller more stable, faster and easy to debug.

The difference between them is that functions must return a value and procedures do not.

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



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

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

A Procedure can be Sub, Function or Property Procedure.

Functions are similar to normal procedures but the main purpose of the functions is to accept certain inputs and pass them on to the main program to finish the execution. They are two types of function, the built-in functions (or internal functions) and the functions created by the programmers.

## 8.1 Function Procedure

Visual Basic has many built-in function. In one respect, function are like miniature programs. They use input, they process the input, and they have output. Some function we encountered earlier are listed in Table 8.1

**Table 8.1:** Some Visual Basic Built-in Function.

| Function | Example            | Input  | Ouput  |
|----------|--------------------|--------|--------|
| Int      | Int(2.6) is 2      | number | number |
| Chr      | Chr(65) is "A"     | number | string |
| Asc      | Asc("Apple") is 65 | string | number |

Format Number (12345.628,1) is 12,345.6 number, number string

Although the input can involve several values, the output always consists of a single value. The items inside the parentheses can be literals (as in Table 5.1), variables, or expressions.

In addition to using built-in function, we can define function of our own. These new function, called function procedure or user-defined function, are defined in much the same way as Sub procedure and are used in the same way as built-in function. Like built-in function, function procedure have a single output that can be of any data type. Function procedure can be used in expressions in exactly the same way as built-in function. Programs refer to them as if they were literals, variables, or expressions. function procedure are defined by function blocks of the form

```
function FunctionName(ByVal var1 As Type1, _
 ByVal var2 As Type2, ...) As DataType
 statement(s)
 Return expression
End function
```

The variables appearing in the top line are called parameters. Variables declared by statements inside the function block have local scope. Function names should be suggestive of the role performed and must conform to the rules for naming variables. The type DataType, which specifies the type of the output, will be one of String, Integer, Double, and so on. In the preceding general code, the next-to-last line specifies the output, which must be of type DataType. Like Sub procedure, function procedure are typed directly into the Code window. (The last line, End function, will appear automatically after the first line is entered into the Code window.)



A variable passed to a function procedure is normally passed by value. It can also be passed by reference and thereby possibly have its value changed by the function procedure. However, passing a variable by reference violates good design principles, since a function is intended to only create a single result and not cause any other changes.

Two examples of function procedure are as follows:

```
function FtoC(ByVal t As Double) As Double
 'Convert Fahrenheit temperature to Celsius
 Return (5/9) * (t - 32)
End function

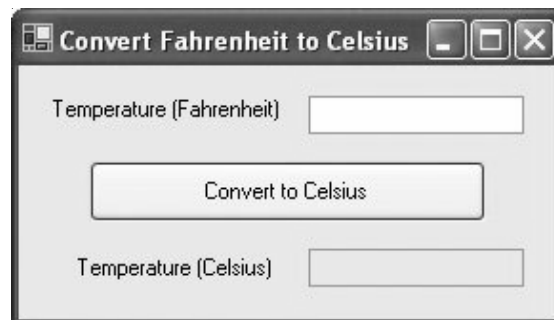
function FirstName(ByVal name As String) As String
 'Extract the first name from a full name
 Dim firstSpace As Integer
 firstSpace = name.IndexOf(" ")
 Return name.Substring(0, firstSpace)
End function
```

The value of each of the preceding function is assigned by a statement of the form `return` expression. The variables *t* and *name* appearing in the preceding function are parameters. They can be replaced with any variable of the same type without affecting the function definition. For instance, the function `FtoC` could have been defined as

```
function FtoC(ByVal temp As Double) As Double
 'Convert Fahrenheit temperature to Celsius
 Return (5/9) * (temp - 32)
End function
```



*Example 1:* The following program uses the function `FtoC`.



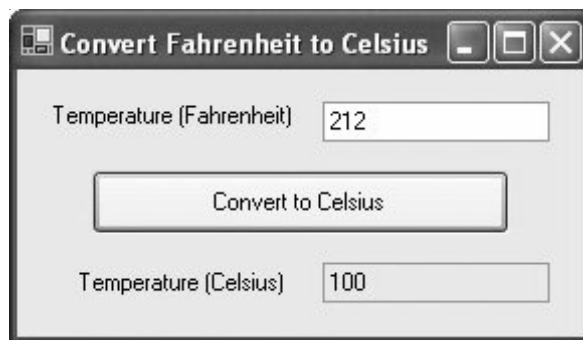
| Object     | Property | Setting                       |
|------------|----------|-------------------------------|
| frmConvert | Text     | Convert Fahrenheit to Celsius |
| lblTempF   | Text     | Temperature (Fahrenheit)      |
| txtTempF   |          |                               |

|            |          |                       |
|------------|----------|-----------------------|
| btnConvert | Text     | Convert to Celsius    |
| lblTempC   | Text     | Temperature (Celsius) |
| txtTempC   | ReadOnly | True                  |

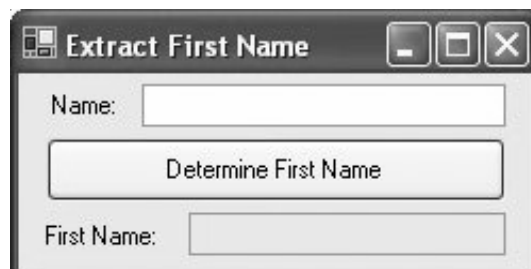
```
Private Sub btnConvert_Click(...) Handles btnConvert.Click
 Dim fahrenheitTemp, celsiusTemp As Double
 fahrenheitTemp = Cdbl(txtTempF.Text)
 celsiusTemp = FtoC(fahrenheitTemp)
 txtTempC.Text = CStr(celsiusTemp)
 'Note: The above four lines can be replaced with the single line
 'txtTempC.Text = CStr(FtoC(Cdbl(txtTempF.Text)))
End Sub

function FtoC(ByVal t As Double) As Double
 'Convert Fahrenheit temperature to Celsius
 Return (5/9) * (t - 32)
End function
```

[Run, type 212 into the text box, and then click the button.]



Example 2: The following program uses the function FirstName.



| Object       | Property | Setting              |
|--------------|----------|----------------------|
| frmFirstName | Text     | Extract First Name   |
| lblName      | Text     | Name                 |
| txtFullName  |          |                      |
| btnDetermine | Text     | Determine First Name |
| txtFirstName | ReadOnly | True                 |

```

Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
 'Determine a person's first name
 Dim name As String
 name = txtFullName.Text
 txtFirstName.Text = FirstName(name)
End Sub

function FirstName(ByVal name As String) As String
 'Extract the first name from a full name
 Dim firstSpace As Integer
 firstSpace = name.IndexOf(" ")
 Return name.Substring(0, firstSpace)
End function

```

[Run, type Thomas Woodrow Wilson into the text box, and then click the button.]



Write a program for Procedures in visual basic.

## 8.2 User-Defined Function Having Several Parameters

The input to a user-defined function can consist of one or more values. Two examples of function with several parameters follow. One-letter variable names have been used so the mathematical formulas will look familiar and be readable. Because the names are not descriptive, the meanings of these variables are carefully spelled out in comment statements.

```

function Hypotenuse(ByVal a As Double, ByVal b As Double) As Double
 'Calculate the hypotenuse of a right triangle

```

```

 'having sides of lengths a and b
 Return Math.Sqrt(a ^ 2 + b ^ 2)
End function

function FutureValue(ByVal p As Double, ByVal r As Double, _
 ByVal c As Double, ByVal n As Double) As Double
 'Find the future value of a bank savings account
 'p principal, the amount deposited
 'r annual rate of interest
 'c number of times interest is compounded per year
 'n number of years
 Dim i As Double 'interest rate per period
 Dim m As Double 'total number of times interest is compounded
 i = r / c
 m = c * n
 Return p * ((1 + i) ^ m)
End function

```



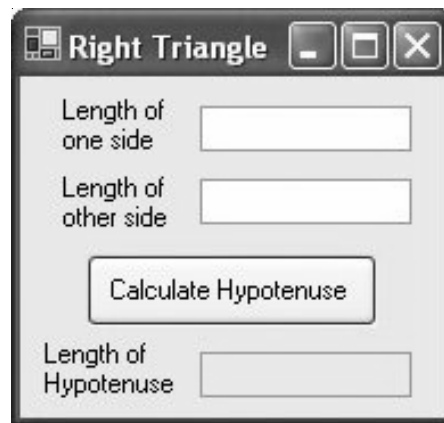
Example 1: The following program uses the Hypotenuse function.

| Object        | Property | Setting              |
|---------------|----------|----------------------|
| frmPythagoras | Text     | Right Triangle       |
| lblSideOne    | AutoSize | False                |
|               | Text     | Length of one side   |
| txtSideOne    |          |                      |
| lblSideTwo    | AutoSize | False                |
|               | Text     | Length of other side |
| txtSideTwo    |          |                      |
| btnCalculate  | Text     | Calculate Hypotenuse |
| lblHyp        | AutoSize | False                |
|               | Text     | Length of Hypotenuse |
| txtHyp        | ReadOnly | True                 |

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
 'Calculate the length of the hypotenuse of a right triangle
 Dim a, b As Double
 a = CDb1(txtSideOne.Text)
 b = CDb1(txtSideTwo.Text)
 txtHyp.Text = CStr(Hypotenuse(a, b))
End Sub

function Hypotenuse(ByVal a As Double, ByVal b As Double) As Double
 'Calculate the hypotenuse of a right triangle
 'having sides of lengths a and b
 Return Math.Sqrt(a ^ 2 + b ^ 2)
End function
```

[Run, type 3 and 4 into the text boxes, and then click the button.]



### **8.3 User-Defined Function Having No Parameters**

Function, like sub function, need not have any parameters.



*Example 1:* The following program uses a parameterless function.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 'Request and display a saying
 txtBox.Text = Saying()
End Sub

function Saying() As String
 'Retrieve a saying from the user
 Return InputBox("What is your favorite saying?")
end function
```

End function

The saying "Less is more." is displayed in the text box.

## 8.4 Comparing Function Procedures with Sub Procedures

Function Procedures differ from Sub Procedures in the way they are accessed. Sub Procedures are invoked with call statements, whereas function are invoked by placing them where you would otherwise expect to find a literal, variable, or expression. Unlike a Function Procedures, a Sub Procedures can't be used in an expression.

Function Procedures can perform the same tasks as Sub Procedures. For instance, they can request input and display text. However, Function Procedures are primarily used to calculate a single value. Normally, Sub Procedures are used to carry out other tasks.



*Did u know?*

The Sub Procedures considered in this book terminate only when End Sub is reached. On the other hand, Function Procedures terminate as soon as the first Return statement is executed. For instance, if a Return statement is followed by a sequence of statements and the Return statement is executed, then the sequence of statements will not be executed.

## 8.5 Collapsing a Procedures with a Region Directive

A group of Procedures or class-level variables can be collapsed behind a captioned rectangle. This task is carried out with a so-called Region directive. To specify a region, precede the code to be collapsed with a line of the form

#Region "Text to be displayed in the rectangle".

and follow the code with the line

#End Region

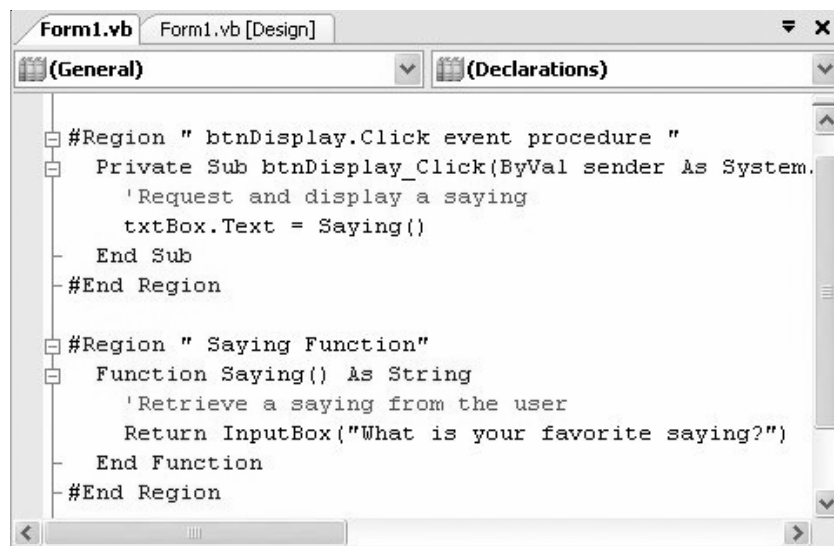


Figure 8.1(a): Region Directives.

A tiny box holding a minus sign will appear to the left of the #Region line. To collapse the code, click on the minus sign. The code will be hidden behind a rectangle captioned with the text you specified and the minus sign will be replaced by a plus sign. Click on the plus sign to expand the region. The Region directive is used to make a program more readable or to create an outline for a program. In Figure 8.1(a), Region directives have been specified for each Procedures in Example . In Figure 8.1(b), these two regions have been collapsed.

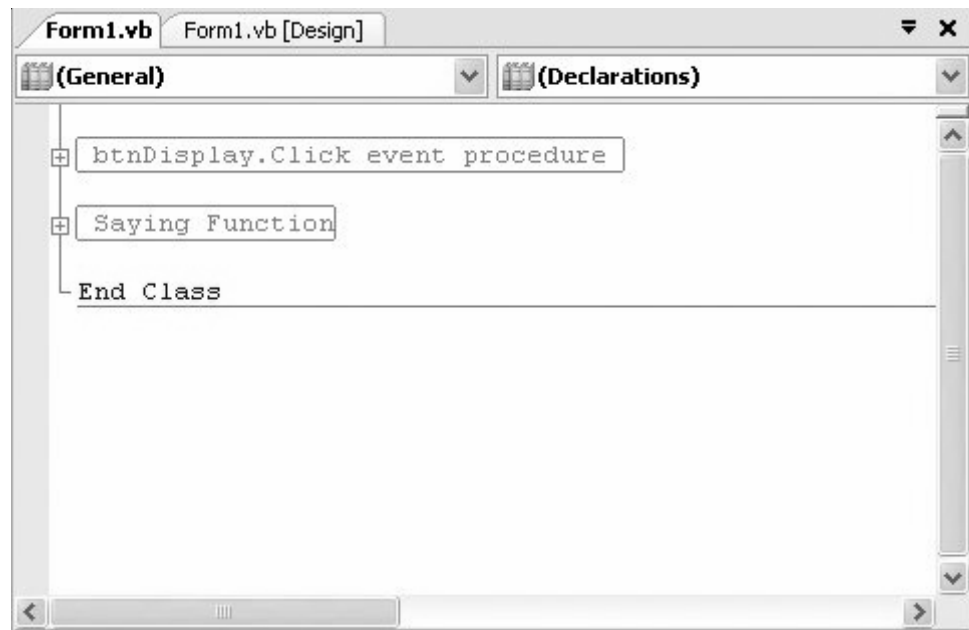


Figure 8.1(b): Collapsed Region.

## 8.6 Modules

Full-featured software usually requires large programs. Writing the code for an event procedure in such a Visual Basic program might pose a complicated problem. One method programmers use to make a complicated problem more understandable is to divide it into smaller, less complex subproblems. Repeatedly using a “divide-and-conquer” approach to break up a large problem into smaller subproblems is called stepwise refinement. Stepwise refinement is part of a larger methodology of writing programs known as top-down design. The term top-down refers to the fact that the more general tasks occur near the top of the design and tasks representing their refinement occur below. Top-down design and structured programming emerged as techniques to enhance programming productivity. Their use leads to programs that are easier to read and maintain. They also produce programs containing fewer initial errors, with these errors being easier to find and correct. When such programs are later modified, there is a much smaller likelihood of introducing new errors.

The goal of top-down design is to break a problem into individual tasks, or modules, that can easily be transcribed into pseudocode, flowcharts, or a program. First, a problem is restated as



several simpler problems depicted as modules. Any modules that remain too complex are broken down further. The process of refining modules continues until the smallest modules can be coded directly. Each stage of refinement adds a more complete specification of what tasks must be performed. The main idea in top-down design is to go from the general to the specific. This process of dividing and organizing a problem into tasks can be pictured using a hierarchy chart. When using top-down design, certain criteria should be met:

1. The design should be easily readable and emphasize small modules size.
2. modules proceed from general to specific as you read down the chart.
3. The modules, as much as possible, should be single minded. That is, they should only perform a single well-defined task.
4. modules should be independent of each other as much as possible, and any relationships among modules should be specified.

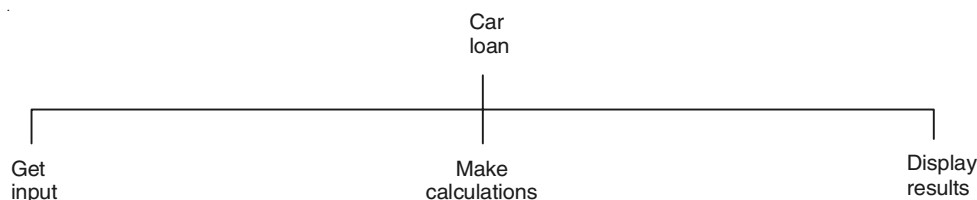


Write a program for using modules in visual basic.

This process is illustrated with the following example:

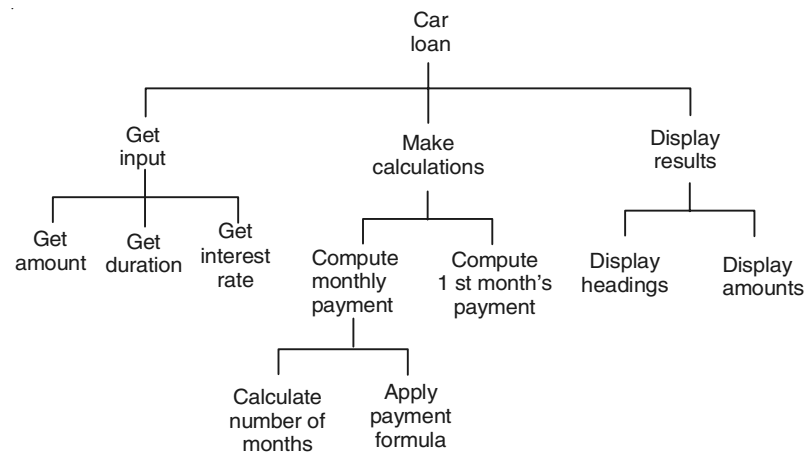


*Example 1:* The chart in Figure 8.2(a) is a hierarchy chart for a program that gives certain information about a car loan. The inputs are the amount of the loan, the duration (in years), and the interest rate. The output consists of the monthly payment and the amount of interest paid during the first month. In the broadest sense, the program calls for obtaining the input, making calculations, and displaying the output. Figure 8.2(a) shows these tasks as the first row of a hierarchy chart.



**Figure 8.2(a):** Beginning of a Hierarchy Chart for the Car Loan Program.

Each of these tasks can be refined into more specific subtasks. (See Figure for the final hierarchy chart.) Most of the subtasks in the third row are straightforward and so do not require further refinement. For instance, the first month's interest is computed by multiplying the amount of the loan by one-twelfth of the annual rate of interest. The most complicated subtask, the computation of the monthly payment, has been broken down further. This task is carried out by applying a standard formula found in finance books; however, the formula requires the number of payments.



**Figure 8.2(b):** Hierarchy Chart for the Car Loan Program.

It is clear from the hierarchy chart that the top modules manipulate the modules beneath them. While the higher-level modules control the flow of the program, the lower-level modules do the actual work. By designing the top modules first, specific processing decisions can be delayed.



#### Case Study

The following program uses the future value function. With the responses shown, the program computes the balance in a savings account when \$100 is deposited for five years at 4% interest compounded quarterly. Interest is earned four times per year at the rate of 1% per interest period. There will be  $4 * 5$ , or 20, interest periods.

**Bank Deposit**

Amount of bank deposit:

Annual rate of interest:

Number of times interest is compounded per year:

Number of years:

Balance:

| Object     | Property | Setting                                             |
|------------|----------|-----------------------------------------------------|
| frmBank    | Text     | Bank Deposit                                        |
| lblAmount  | Text     | Amount of bank deposit:                             |
| txtAmount  |          |                                                     |
| lblRate    | Text     | Annual rate of interest:                            |
| txtRate    |          |                                                     |
| lblNumComp | AutoSize | False                                               |
|            | Text     | Number of times interest<br>is compounded per year: |
| txtNumComp |          |                                                     |
| lblNumYrs  | Text     | Number of years:                                    |
| txtNumYrs  |          |                                                     |
| btnCompute | Text     | Compute Balance                                     |
| lblBalance | Text     | Balance:                                            |
| txtBalance | ReadOnly | True                                                |

```

Private Sub btnCompute_Click(...) Handles btnCompute.Click
 'Find the future value of a bank deposit
 Dim p As Double 'principal, the amount deposited
 Dim r As Double 'annual rate of interest
 Dim c As Double 'number of times interest is compounded per year
 Dim n As Double 'number of years
 InputData(p, r, c, n)
 DisplayBalance(p, r, c, n)
End Sub

Sub InputData(ByRef p As Double, ByRef r As Double, _
 ByRef c As Double, ByRef n As Double)
 'Get the four values from the text boxes
 p = Cdbl(txtAmount.Text)
 r = Cdbl(txtRate.Text)
 c = Cdbl(txtNumComp.Text)
 n = Cdbl(txtNumYrs.Text)
End Sub

Sub DisplayBalance(ByVal p As Double, ByVal r As Double, _

```

```
 ByVal c As Double, ByVal n As Double)
'Display the balance in a text box
Dim balance As Double
balance = FutureValue(p, r, c, n)
txtbalance.Text = FormatCurrency(balance)
End Sub

function FutureValue(ByVal p As Double, ByVal r As Double, _
 ByVal c As Double, ByVal n As Double) As Double
'Find the future value of a bank savings account
'p principal, the amount deposited
'r annual rate of interest
'c number of times interest is compounded per year
'n number of years
Dim i As Double 'interest rate per period
Dim m As Double 'total number of times interest is compounded
i = r / c
m = c * n
Return p * ((1 + i) ^ m)
End function
```

[Run, type 100, .04, 4, and 5 into the text boxes, then click the button.]



### Questions

1. What is the difference between MsgBox Statement and MsgBoxQ function?
2. Write a visual Basic program for modules.

## 8.7 Summary

- User defined function can consists of one or more variables.
- Comparing function procedures with sub-procedures are invoked with call statements.
- Top-down design and structured programming emerged as techniques to enhance programming productivity.

## 8.8 Keywords

**Function procedure:** In addition to using built-in function, we can define function of our own. These new function, called function procedure or user-defined function.

**Future value function:** The value of an asset or cash at a specified date in the future that is equivalent in value to a specified sum today.

**Parameters:** The variables appearing in the top line are called parameters.

**Region directive:** The Region directive is used to make a program more readable or to create an outline for a program.

**Stepwise refinement:** Repeatedly using a “divide-and-conquer” approach Repeatedly using a “divide-and-conquer” approach.



### Lab Exercise

1. Suppose a program contains the lines

```
Dim n As Double, x As String
```

```
lstOutput.Items.Add(Arc(n, x))
```

What types of inputs (numeric or string) and output does the function Arc have?

2. Suppose a fixed amount of money is deposited at the beginning of each month into a savings account paying 6% interest compounded monthly. After each deposit is made, [new balance] = 1.005 \* [previous balance one month ago] + [fixed amount]. Write a program that requests the fixed amount of the deposits as input and displays the balance after each of the first four deposits. A sample outcome when 800 is typed into the text box for the amount deposited each month follows.

|         |          |
|---------|----------|
| Month 1 | 800.00   |
| Month 2 | 1,604.00 |
| Month 3 | 2,412.02 |
| Month 4 | 3,224.08 |

## **8.9 Self Assessment Questions**

1. You have created a new property procedure in a class module. You have not specified its Scope. What scope is it?
  - (a) Private
  - (b) Friend
  - (c) Public
  - (d) Global
  - (e) None
2. What is the syntax to assign the return value for a Property Get procedure named LastName, where LastName is a String?
  - (a) Set LastName = ReturnValue
  - (b) Return ReturnValue
  - (c) Exit ReturnValue
  - (d) LastName = ReturnValue
3. How do you implement callback functionality from a class to its calling code in a standard executable?
  - (a) Provide a custom class event with at least one Boolean By Val parameter.
  - (b) Provide a custom class event as a function with a Boolean return value.
  - (c) Provide a custom class event with at least one By Reference parameter.
  - (d) There is no way to provide callback functionality from a class in a standard executable.
4. When does an event procedure for a class's custom event appear in the code window of a VB application?
  - (a) You must manually declare the event procedure.
  - (b) After you write a declaration of the event in the class.
  - (c) After you use WithEvents to declare an object variable of the class.
  - (d) After you write code in the class to raise the event.
5. You are designing a class module that will be used as a standalone EXE server. Which of the following is true about the Instancing property for the class?
  - (a) The class must be defined Private.
  - (b) A PublicNotCreatable class must have another class that provides access to it.
  - (c) A program using a class defined as GlobalMultiUse needs to explicitly mention the class name to use its properties and methods.
  - (d) The class can be defined as SingleUse.
6. A module-level variable is visible to a watch of which type of scope?
  - (a) Global
  - (b) Module-level

- (c) Procedure-level
  - (d) Unbound
7. The highest Watch scope meaningful to a variable declared in a module using the Private keyword is
- (a) Global
  - (b) Module-level
  - (c) Procedure-level
  - (d) Indeterminate
8. Which statement is true about Record Maintenance?
- (a) This utility program actually creates new tables in the Solomon database.
  - (b) This utility is used to define the following properties of a table: Name, Fields and Indexes.
  - (c) This utility allows a developer to define the names of new tables. It does not actually create tables in the Solomon database.
  - (d) This utility reads the actual database schema from the Solomon database and displays all actual tables in that database except for MS SQL system tables.
9. In building your class module, you've been asked to make a property write-only. How can you specify a write-only property for a class?
- (a) Define a Property Set procedure without a Property Let procedure.
  - (b) Define a Property Let procedure without a Property Set procedure.
  - (c) Define a Property Let procedure without a Property Get procedure.
  - (d) You cannot make a property write-only in a Visual Basic class module.

### **8.10 Review Questions**

1. To determine the number of square centimeters of tin needed to make a tin can, add the square of the radius of the can to the product of the radius and height of the can, and then multiply this sum by 6.283. Write a program that requests the radius and height of a tin can in centimeters as input and displays the number of square centimeters required to make the tin can.
2. According to Plato, a man should marry a woman whose age is half his age plus seven years. Write a program that requests a man's age as input and gives the ideal age of his wife.
3. The federal government developed the body mass index (BMI) to determine ideal weights. Body mass index is calculated as 703 times the weight in pounds, divided by the square of the height in inches, and then rounded to the nearest whole number. Write a program that accepts a person's weight and height as input and gives the person's body mass index.  
**Note:** A BMI of 19 to 25 corresponds to a healthy weight.
4. In order for exercise to be beneficial to the cardiovascular system, the heart rate (number of heart beats per minute) must exceed a value called the training heart rate, THR. A

person's THR can be calculated from his age and resting heart rate (pulse when first awakening) as follows:

- (a) Calculate the maximum heart rate as  $220 - \text{age}$ .
- (b) Subtract the resting heart rate from the maximum heart rate.
- (c) Multiply the result in step (b) by 60%, and then add the resting heart rate.

Write a program to request a person's age and resting heart rate as input and display their THR. (Test the program with an age of 20 and a resting heart rate of 70, and then determine your training heart rate.)

5. The three ingredients for a serving of popcorn at a movie theater are popcorn, butter substitute, and a bucket. Write a program that requests the cost of these three items and the price of the serving as input and then displays the profit. (Test the program where popcorn costs 5 cents, butter substitute costs 2 cents, the bucket costs 25 cents, and the selling price is \$5.)
6. Rewrite the population-density program from Example 4 of Section 4.1 using a function to calculate the population density.
7. The original cost of airmail letters was 5 cents for the first ounce and 10 cents for each additional ounce. Write a program to compute the cost of a letter whose weight is given by the user in a text box. Use a function called Ceil that rounds noninteger numbers up to the next integer. The function Ceil can be defined by  $\text{Ceil}(x) = \text{Int}(x) + 1$ . (Test the program with the weights 4, 1, 2.5, and .5 ounces.)
8. Suppose a fixed amount of money is deposited at the beginning of each month into a savings account paying 6% interest compounded monthly. After each deposit is made, [new balance] = 1.005 \* [previous balance one month ago] + [fixed amount]. Write a program that requests the fixed amount of the deposits as input and displays the balance after each of the first four deposits. A sample outcome when 800 is typed into the text box for the amount deposited each month follows.

Month 1      800.00

Month 2    1,604.00

Month 3    2,412.02

Month 4    3,224.08

9. Write a program to request the name of a United States senator as input and display the address and greeting for a letter to the senator. Assume the name has two parts, and use a function to determine the senator's last name. A sample outcome when Robert Smith is typed into the input dialog box requesting the senator's name follows.

The Honorable Robert Smith

United States Senate

Washington, DC 20001

Dear Senator Smith,

10. Difference between Class Module and Standard Module.



11. Define the scope of Public, Private, Friend procedures.
12. Difference between a function and a subroutine, Dynaset and Snapshot, early and late binding, image and picture controls, linked object and embedded Object, listbox and combo box, Listindex and Tab index, modal and modeless window, Object and Class, query unload and unload in form, declaration and instantiation of an object.
13. What is a module? Give different types of module with examples.
14. Write a program that requests three scores as input and displays the average of the two highest scores. The input and output should be handled by Sub procedures, and the average should be determined by a user-defined function.
15. You want to create a read-only property. How would you define the Property procedure?
16. Suppose a program contains the lines

```
Dim n As Double, x As String
```

```
lstOutput.Items.Add(Arc(n, x))
```

What types of inputs (numeric or string) and output does the function Arc have?

17. What is displayed in the text box when btnCompute is clicked?

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
 'How many gallons of apple cider can we make?
 Dim gallonsPerBushel, apples As Double
 GetData(gallonsPerBushel, apples)
 DisplayNumOfGallons(gallonsPerBushel, apples)
End Sub

function Cider(ByVal g As Double, ByVal x As Double) As Double
 Return g * x
End function

Sub DisplayNumOfGallons(ByVal galPerBu As Double, _
 ByVal apples As Double)
 txtOutput.Text = "You can make "& Cider(galPerBu, apples) _
 & " gallons of cider."
End Sub

Sub GetData(ByRef gallonsPerBushel As Double, _
 ByRef apples As Double)
 'gallonsPerBushel Number of gallons of cider one bushel
 'of apples makes
 'apples Number of bushels of apples available
 gallonsPerBushel = 3
 apples = 9
End Sub
```

### Answers for Self Assessment Questions

- |        |              |        |        |
|--------|--------------|--------|--------|
| 1. (c) | 2. (d)       | 3. (c) | 4. (c) |
| 5. (d) | 6. (b and c) | 7. (b) | 8. (c) |
| 9. (c) |              |        |        |

### 8.11 Further Readings



*Books*

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net."



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 9: Menus and Dialog Boxes**

### **CONTENTS**

Objectives

Introduction

9.1 Designing Menus Using Menu Editor

9.1.1 To Display the Menu Editor

9.1.2 Using the list Box in the Menu Editor

9.1.3 To Create Menu Controls in the Menu Editor

9.2 Assigning Access Keys and Shortcut Keys

9.2.1 Access keys

9.2.2 Shortcut Keys

9.3 Separating Menu Items

9.3.1 To Create a Separator Bar in the Menu Editor

9.4 Creating Pop-up Menus

9.4.1 The Flags Argument

9.4.2 The Boldcommand Argument

9.5 Controlling Menus at Run Time

9.5.1 Enabling and Disabling Menu Commands

9.5.2 Displaying a Check Mark on a Menu Control

9.5.3 Marking Menu Controls Invisible

9.5.4 Adding Menu Controls at Run Time

9.6 Modal and Modeless Dialog Boxes

9.6.1 To Display a Form as a Modal Dialog Box

9.6.2 To Display a Form as a Modeless Dialog Box

9.6.3 To Display a Form as a Child of Another Form

9.7 Using Predefined Dialog Boxes

9.7.1 Prompting for Input with Input Box

9.7.2 Displaying Information with MsgBox

9.8 Summary

9.9 Keywords

9.10 Self Assessment Questions

9.11 Review Questions

9.12 Further Readings

## Objectives

*After studying this unit, you will be able to:*

- Discuss designing menus using menu editor
- Explain Assigning access keys and shortcut keys
- Understand the separating menu items
- Discuss the controlling menus at run time
- Explain model and modeless dialog boxes
- Understand predefined dialog boxes uses

## Introduction

Menus in Visual basic 6.0 are created using the Menu object defined in the VB Library. The menu created so are flat Win98 style menus and do not support icons. Further they cannot be created at runtime and require “Menu Editor”, modification though can be made during runtime.

Visual Basic Editor (VBE) and create a custom dialog box. Creating your own dialog boxes, or User Forms as they are officially called, we can save a lot of time when we need to acquire information from users. Rather than displaying a series of input boxes to gather information and then a series of message boxes to instruct the user, a custom dialog box can do a lot of the work for us. And they’re not as hard to create and use as you might think. Custom dialog boxes are not only useful for gathering information such as the user’s name or personal data. A user recently wanted to create a series of tables, each table with a slightly varied format. He could have recorded one macro for each type of table but a more efficient way would be to create a custom dialog box that gathers optional information. Then that information could be transferred into the various format settings so that one table building macro could create several different types of tables.

## 9.1 Designing Menus Using Menu Editor

We can use the Menu Editor to create new menus and menu bars, add new commands to existing menus, replace existing menu commands with your own commands, and change and delete existing menus and menu bars.

### 9.1.1 To Display the Menu Editor

- From the **Tools** menu, choose **Menu Editor** –or– Click the **Menu Editor** button on the toolbar.

This opens the Menu Editor, shown in Figure 9.1.

While most menu control properties can be set using the Menu Editor, all menu properties are available in the Properties window. The two most important properties for menu controls are:

- **Name** — This is the name you use to reference the menu control from code.
- **Caption** — This is the text that appears on the control.

Other properties in the Menu Editor, including Index, Checked, and NegotiatePosition, are described later in this unit.

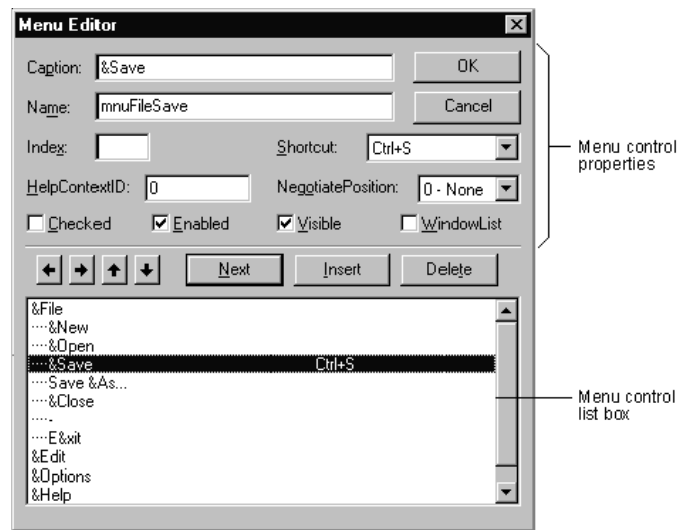


Figure 9.1: The Menu Editor.

### 9.1.2 Using the List Box in the Menu Editor

The menu control list box (the lower portion of the Menu Editor) lists all the menu controls for the current form. When you type a menu item in the Caption text box, that item also appears in the menu control list box. Selecting an existing menu control from the list box allows you to edit the properties for that control.

For example, Figure 6.1 shows the menu controls for a File menu in a typical application. The position of the menu control in the menu control list box determines whether the control is a menu title, menu item, submenu title, or submenu item:

- A menu control that appears flush left in the list box is displayed on the menu bar as a menu title.
- A menu control that is indented once in the list box is displayed on the menu when the user clicks the preceding menu title.
- An indented menu control followed by menu controls that are further indented becomes a submenu title. Menu controls indented below the submenu title become items of that submenu.
- A menu control with a hyphen (-) as its Caption property setting appears as a separator bar. A separator bar divides menu items into logical groups.



Notes

A menu control cannot be a separator bar if it is a menu title, has submenu items, is checked or disabled, or has a shortcut key.

### 9.1.3 To Create Menu Controls in the Menu Editor

1. Select the form.
2. From the **Tools** menu, choose **Menu Editor** –or– Click the **Menu Editor** button on the toolbar.
3. In the **Caption** text box, type the text for the first menu title that you want to appear on the menu bar. Also, place an ampersand (&) before the letter you want to be the access key

for that menu item. This letter will automatically be underlined in the menu. The menu title text is displayed in the menu control list box.

4. In the **Name** text box, type the name that you will use to refer to the menu control in code. See “Menu Title and Naming Guidelines” later in this chapter.
5. Click the left arrow or right arrow buttons to change the indentation level of the control.
6. Set other properties for the control, if you choose. You can do this in the Menu Editor or later, in the Properties window.
7. Choose **Next** to create another menu control –or– Click **Insert** to add a menu control between existing controls. You can also click the up arrow and down arrow buttons to move the control among the existing menu controls.
8. Choose **OK** to close the Menu Editor when you have created all the menu controls for that form.

The menu titles you create are displayed on the form. At design time, click a menu title to drop down its corresponding menu items.

## 9.2 Assigning Access Keys and Shortcut Keys

You can improve keyboard access to menu commands by defining access keys and shortcut keys.

### 9.2.1 Access Keys

**Access keys** allow the user to open a menu by pressing the ALT key and typing a designated letter. Once a menu is open, the user can choose a control by pressing the letter (the access key) assigned to it. For example, ALT+E might open the Edit menu, and P might select the Paste menu item. An access-key assignment appears as an underlined letter in the menu control’s caption, as shown in Figure 9.2.

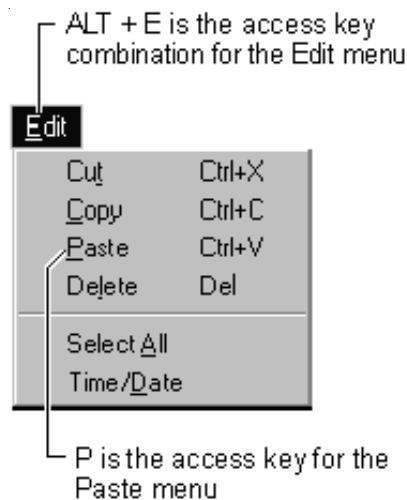


Figure 9.2: Access Keys.

#### 9.2.1.1 To Assign an Access Key to a Menu Control in the Menu Editor

1. Select the menu item to which you want to assign an access key.
2. In the **Caption** box, type an ampersand (&) immediately in front of the letter you want to be the access key.

For example, if the Edit menu shown in Figure 9.2 is open, the following Caption property settings respond to the corresponding keys.

| Menu control caption | Caption property | Access keys |
|----------------------|------------------|-------------|
| Cut                  | Cu&t             | t           |
| Copy                 | C&opy            | o           |
| Paste                | &Paste           | p           |
| Delete               | De&lete          | l           |
| Select All           | Select &All      | a           |
| Time/Date            | Time/&Date       | d           |



Do not use duplicate access keys on menus. If you use the same access key for more than one menu item, the key will not work. For example, if C is the access key for both Cut and Copy, when you select the Edit menu and press C, the Copy command will be selected, but the application will not carry out the command until the user presses ENTER. The Cut command will not be selected at all.

## 9.2.2 Shortcut Keys

**Shortcut keys** run a menu item immediately when pressed. Frequently used menu items may be assigned a keyboard shortcut, which provides a single-step method of keyboard access, rather than a three-step method of pressing ALT, a menu title access character, and then a menu item access character. Shortcut key assignments include function key and control key combinations, such as CTRL+F1 or CTRL+A. They appear on the menu to the right of the corresponding menu item, as shown in Figure 9.3.

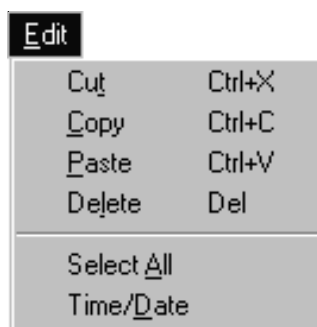


Figure 9.3: Shortcut Keys.

### 9.2.2.1 To Assign a Shortcut Key to a Menu Item

1. Open the **Menu Editor**.
2. Select the menu item.
3. Select a function key or key combination in the **Shortcut** combo box.

To remove a shortcut key assignment, choose “(none)” from the top of the list.



Notes

Shortcut keys appear automatically on the menu; therefore, you do not have to enter CTRL+key in the Caption box of the Menu Editor.

### 9.3 Separating Menu Items

A separator bar is displayed as a horizontal line between items on a menu. On a menu with many items, you can use a separator bar to divide items into logical groups. For example, the File menu in Visual Basic uses separator bars to divide its menu items into three groups, as shown in Figure 9.4.

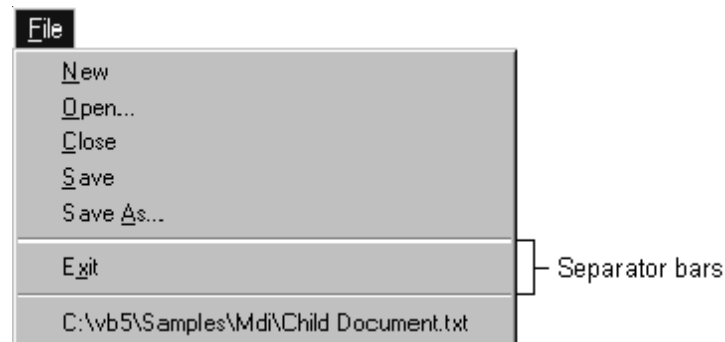


Figure 9.4: Separator Bars.

#### 9.3.1 To Create a Separator Bar in the Menu Editor

1. If you are adding a separator bar to an existing menu, choose **Insert** to insert a menu control between the menu items you want to separate.
2. If necessary, click the right arrow button to indent the new menu item to the same level as the menu items it will separate.
3. Type a hyphen (-) in the **Caption** text box.
4. Set the **Name** property.
5. Choose **OK** to close the Menu Editor.



Notes

Although separator bars are created as menu controls, they do not respond to the Click event, and users cannot choose them.

### 9.4 Creating Pop-up Menus

A pop-up menu is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the pop-up menu depend on where the pointer was located when the right mouse button was pressed; therefore, pop-up menus are also called context menus. In Microsoft Windows 95 or later systems, we activate context menus by clicking the right mouse button.





Figure 9.5: Pop-up Menu.

Any menu that has at least one menu item can be displayed at run time as a pop-up menu. To display a pop-up menu, use the `PopupMenu` method. This method uses the following syntax:

```
[object.]PopupMenu menuname [, flags [,x [, y [, boldcommand]
```

For example, the following code displays a menu named `mnuFile` when the user clicks a form with the right mouse button. You can use the `MouseUp` or `MouseDown` event to detect when the user clicks the right mouse button, although the standard is to use the `MouseUp` event:

```
Private Sub Form_MouseUp (Button As Integer, Shift As _
 Integer, X As Single, Y As Single)
 If Button = 2 Then ' Check if right mouse button
 ' was clicked.
 PopupMenu mnuFile ' Display the File menu as a
 ' pop-up menu.
 End If
End Sub
```

Any code following a call to the `PopupMenu` method is not run until the user selects an item in the menu or cancels the menu.



Notes

Only one pop-up menu can be displayed at a time. While a pop-up menu is displayed, calls to the `PopupMenu` method are ignored. Calls to the `PopupMenu` method are also ignored whenever a menu control is active.

Often we want a pop-up menu to access options that are not usually available on the menu bar. To create a menu that will not display on the menu bar, make the top-level menu item invisible at design time (make sure the `Visible` check box in the Menu Editor is not checked). When Visual Basic displays a pop-up menu, the `visible` property of the specified top-level menu is ignored.

### 9.4.1 The Flags Argument

We use the flags argument in the PopupMenu method to further define the location and behavior of a pop-up menu. The following table lists the flags available to describe a pop-up menu's location.

**Table 9.1:** Flags Available to Describe a Pop-up Menu's Location

| Location constants     | Description                                                                 |
|------------------------|-----------------------------------------------------------------------------|
| vbPopupMenuLeftAlign   | Default. The specified x location defines the left edge of the pop-up menu. |
| vbPopupMenuCenterAlign | The pop-up menu is centered around the specified x location.                |
| vbPopupMenuRightAlign  | The specified x location defines the right edge of the pop-up menu.         |

The following table lists the flags available to describe a pop-up menu's behavior.

**Table 9.2:** Flags Available to Describe a Pop-up Menu's Behavior

| Behavior constants     | Description                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------|
| vbPopupMenuLeftButton  | Default. The pop-up menu is displayed when the user clicks a menu item with the left mouse button only.   |
| vbPopupMenuRightButton | The pop-up menu is displayed when the user clicks a menu item with either the right or left mouse button. |

To specify a flag, we combine one constant from each group using the Or operator. The following code displays a pop-up menu with its top border centered on a form when the user clicks a command button. The pop-up menu triggers Click events for menu items that are clicked with either the right or left mouse button.

```
Private Sub Command1_Click ()
 ' Dimension X and Y variables.
 Dim xloc, yloc

 ' Set X and Y variables to center of form.
 xloc = ScaleWidth / 2
 yloc = ScaleHeight / 2

 ' Display the pop-up menu.
 PopupMenu mnuEdit, vbPopupMenuCenterAlign Or _
 vbPopupMenuRightButton, xloc, yloc
End Sub
```

### 9.4.2 The Boldcommand Argument

We use the `boldcommand` argument to specify the name of a menu control in the displayed pop-up menu that you want to appear in bold. Only one menu control in the pop-up menu can be bold.

## 9.5 Controlling Menus at Run Time

The menus you create at design time can also respond dynamically to run-time conditions. For example, if a menu item action becomes inappropriate at some point, we can prevent users from selecting that menu item by disabling it. In the MDI NotePad application, for example, if the clipboard doesn't contain any text, the Paste menu item is dimmed on the Edit menu, and users cannot select it.

We can also dynamically add menu items, if you have a menu control array. This is described in "Adding Menu Controls at Run Time," later in this topic.

We can also program your application to use a check mark to indicate which of several commands was last selected. For example, the Options, Toolbar menu item from the MDI NotePad application displays a check mark if the toolbar is displayed. Other menu control features described in this section include code that makes a menu item visible or invisible and that adds or deletes menu items.

### 9.5.1 Enabling and Disabling Menu Commands

All menu controls have an `Enabled` property, and when this property is set to `False`, the menu is disabled and does not respond to user actions. Shortcut key access is also disabled when `Enabled` is set to `False`. A disabled menu control appears dimmed, like the Paste menu item in Figure 9.6.

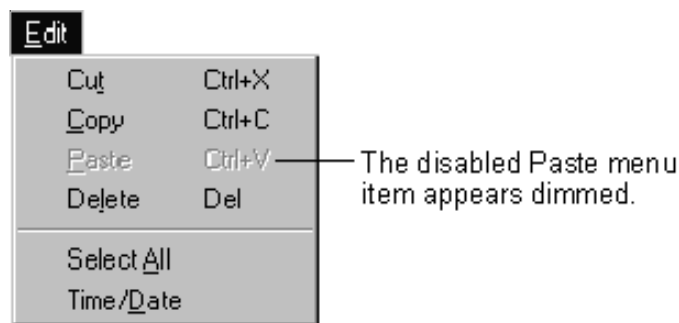


Figure 9.6: A Disabled Menu Item.

For example, this statement disables the Paste menu item on the Edit menu of the MDI NotePad application:

```
mnuEditPaste.Enabled = False
```

Disabling a menu title in effect disables the entire menu, because the user cannot access any menu item without first clicking the menu title. For example, the following code would disable the Edit menu of the MDI Notepad application:

```
mnuEdit.Enabled = False
```

### 9.5.2 Displaying a Check Mark on a Menu Control

Using the Checked property, we can place a check mark on a menu to:

- Tell the user the status of an on/off condition. Choosing the menu command alternately adds and removes the check mark.
- Indicate which of several modes is in effect. The Options menu of the MDI Notepad application uses a check mark to indicate the state of the toolbar, as shown in Figure 9.7.



Figure 9.7: A Checked Menu Item.

We create check marks in Visual Basic with the Checked property. Set the initial value of the Checked property in the Menu Editor by selecting the check box labeled Checked. To add or remove a check mark from a menu control at run time, set its Checked property from code. For example:

```
Private Sub mnuOptions_Click ()
 ' Set the state of the check mark based on
 ' the Visible property.
 mnuOptionsToolbar.Checked = picToolbar.Visible
End Sub
```

### 9.5.3 Making Menu Controls Invisible

In the Menu Editor, you set the initial value of the Visible property for a menu control by selecting the check box labeled Visible. To make a menu control visible or invisible at run time, set its Visible property from code. For example:

```
mnuFileArray(0).Visible = True ' Make the control
 ' visible.

mnuFileArray(0).Visible = False ' Make the control
 ' invisible.
```

When a menu control is invisible, the rest of the controls in the menu move up to fill the empty space. If the control is on the menu bar, the rest of the controls on the menu bar move left to fill the space.



Notes

Making a menu control invisible effectively disables it, because the control is inaccessible from the menu, access or shortcut keys. If the menu title is invisible, all the controls on that menu are unavailable.

### 9.5.4 Adding Menu Controls at Run Time

A menu can grow at run time. In Figure 9.8, for example, as files are opened in the SDI NotePad application, menu items are dynamically created to display the path names of the most recently opened files.

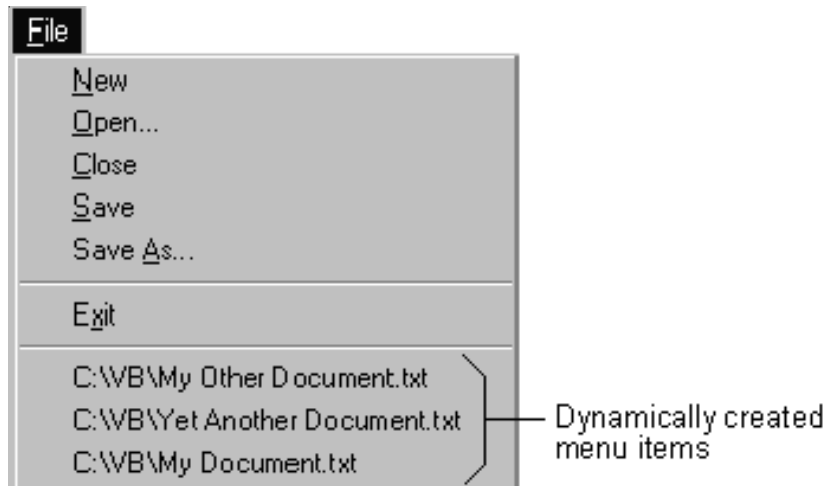


Figure 9.8: Menu Control array Elements Created and Displayed at Run Time.

We must use a control array to create a control at run time. Because the `mnuRecentFile` menu control is assigned a value for the `Index` property at design time, it automatically becomes an element of a control array — even though no other elements have yet been created. When we create `mnuRecentFile(0)`, we actually create a separator bar that is invisible at run time. The first time a user saves a file at run time, the separator bar becomes visible, and the first file name is added to the menu. Each time you save a file at run time, additional menu controls are loaded into the array, making the menu grow. Controls created at run time can be hidden by using the `Hide` method or by setting the control's `Visible` property to `False`. If we want to remove a control in a control array from memory, we will use the `Unload` statement.

### 9.6 Modal and Modeless Dialog Boxes

Dialog boxes are either modal or modeless. A **modal** dialog box must be closed (hidden or unloaded) before you can continue working with the rest of the application. For example, a dialog box is modal if it requires you to click OK or Cancel before you can switch to another form or dialog box.

The About dialog box in Visual Basic is modal. Dialog boxes that display important messages should always be modal — that is, the user should always be required to close the dialog box or respond to its message before proceeding.

**Modeless** dialog boxes let you shift the focus between the dialog box and another form without having to close the dialog box. You can continue to work elsewhere in the current application while the dialog box is displayed. Modeless dialog boxes are rare. From the Edit menu, the Find dialog box in Visual Basic is an example of a modeless dialog box. Use modeless dialog boxes to display frequently used commands or information.

### 9.6.1 To Display a Form as a Modal Dialog Box

- Use the Show method with a style argument of vbModal (a constant for the value 1).



*Example:* \ Display frmAbout as a modal dialog.  
 frmAbout.Show vbModal

### 9.6.2 To Display a Form as a Modeless Dialog Box

- Use the Show method without a style argument.



*Example:* \ Display frmAbout as a modeless dialog.  
 frmAbout.Show



Notes

If a form is displayed as modal, the code following the Show method is not executed until the dialog box is closed. However, when a form is shown as modeless, the code following the Show method is executed immediately after the form is displayed.

The Show method has another optional argument, owner, that can be used to specify a parent-child relationship for a form. we can pass the name of a form to this argument to make that form the owner of the new form.

### 9.6.3 To Display a Form as a Child of Another Form

- Use the Show method with both style and owner arguments.



*Example:* \ Display frmAbout as a modeless child of frmMain.  
 frmAbout.Show vbModeless, frmMain

Using the owner argument with the Show method ensures that the dialog box will be minimized when it's parent is minimized, or unloaded should the parent form be closed.

## 9.7 Predefined Dialog Boxes

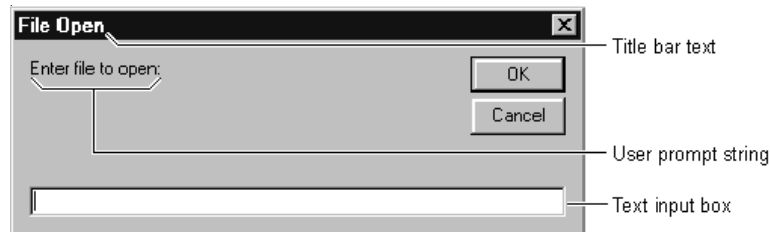
The easiest way to add a dialog box to your application is to use a predefined dialog, because you don't have to worry about designing, loading, or showing the dialog box. However, your control over its appearance is limited. Predefined dialog boxes are always modal. The following table lists the functions you can use to add predefined dialog boxes to your Visual Basic application.

**Table 9.3:** Functions to Add Predefined Dialog

| Use this function | To do this                                                                                                   |
|-------------------|--------------------------------------------------------------------------------------------------------------|
| InputBox function | Display a command prompt in a dialog box, and return whatever is entered by the user.                        |
| MsgBox function   | Display a message in a dialog box, and return a value indicating the command button was clicked by the user. |

### 9.7.1 Prompting for Input with InputBox

Use the InputBox function to solicit data from the user. This function displays a modal dialog box that asks the user to enter some data. The text input box shown in Figure 9.9 prompts the user for the name of the file to open.



**Figure 9.9:** A Dialog Box Using the InputBox Function.

The following code displays the input box shown in Figure 9.9:

```
FileName = InputBox("Enter file to open:", "File Open")
```



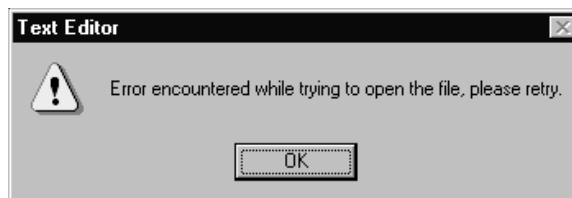
Notes

Remember that when you use the InputBox function, you have little control over the components of the dialog box. You can change only the text in the title bar, the command prompt displayed to the user, the position of the dialog box on the screen, and whether or not it displays a Help button.

### 6.7.2 Displaying Information with MsgBox

Use the MsgBox function to get yes or no responses from users, and to display brief messages, such as errors, warnings, or alerts in a dialog box. After reading the message, the user chooses a button to close the dialog box.

An application named Text Editor might display the message dialog box shown in Figure 9.10 if a file cannot be opened.



**Figure 9.10:** An Error Message Dialog Box Created Using the MsgBox Function.

The following code displays the message box shown in Figure 9.10:

```
MsgBox "Error encountered while trying to open file, _
please retry.", vbExclamation, "Text Editor"
```



Notes

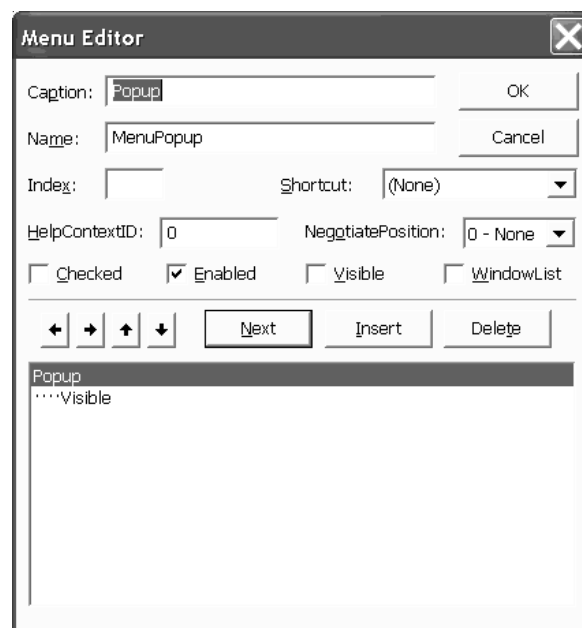
Modality can either be limited to the application or the system. If a message box's modality is limited to the application (default), then users cannot switch to another part of the application until the dialog box is dismissed, but they can switch to another application. A system modal message box does not allow the user to switch to another application until the message box is dismissed.



### Case Study

#### Write a program Creating Menus with the Menu Editor

Menus are added to Windows applications using the Tool, Menu Editor (Ctrl+E shortcut) in VB6. The Menu Editor allows you to create menus by typing the Caption and Name of each menu item and initialize menu items by setting properties for each menu item in the Menu Editor dialog box (see Figure 9.11). Referring to figure one, the menu item that has the focus is the item selected in the list in the bottom half of the Menu Editor dialog. Menu items at the extreme left edge are top-level menu items and each level of indentation represents an additional level of nesting.



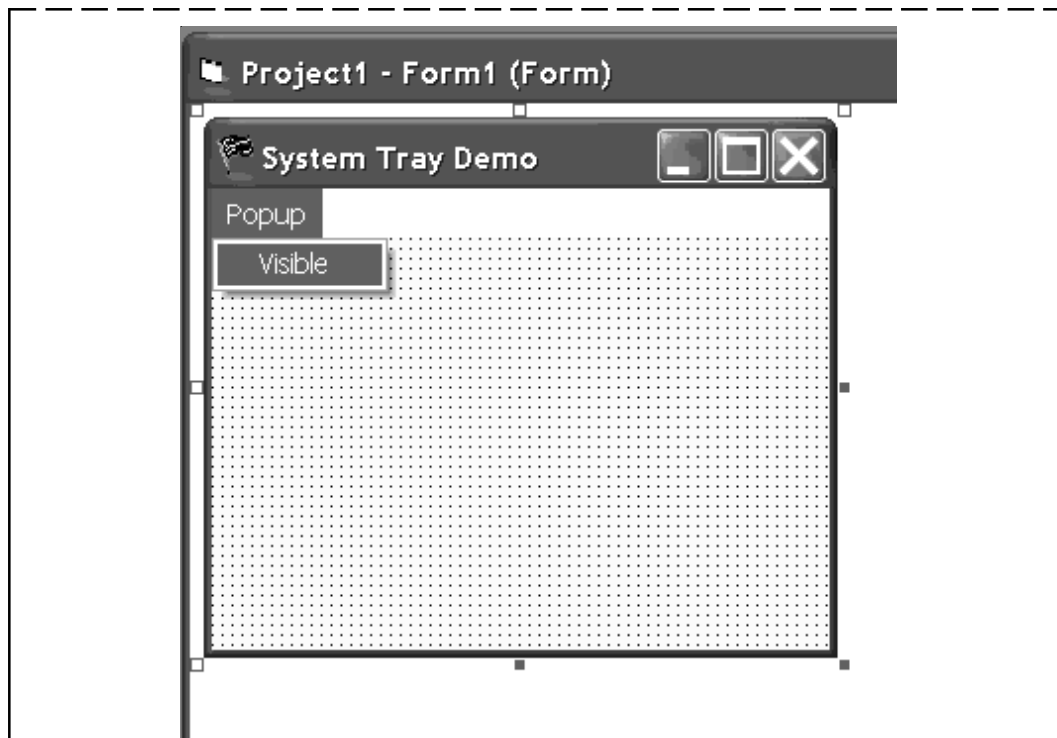
**Figure 9.11:** The Menu Editor is Used to Design Menus in VB6.

From Figure 9.11 we can see that a top-level menu named Popup is selected. The caption that will be displayed is the value in the Caption field, in the example this is "Popup". If we were to add an ampersand to the value of the Caption property immediately before the first P in popup—&Popup—then the first P in popup would be the hotkey for the Popup menu. When the menu is displayed it would appear as follows: Popup. Further if the hotkey P were used then pressing Alt+P when the application containing the popup menu were run would cause the Popup menu to be displayed.

Other properties for the Menu Editor are pretty intuitive. You can use the VB6 help if you are unfamiliar with other Menu Editor properties.

If we were to check the Visible property for the menu shown then the Popup menu would appear as shown in Figure 9.12 (when displayed).





**Figure 9.12:** The Pop-up Menu Described in the Menu Editor in Figure 6.11 Shown in Design Mode Dropped Down.

Menus that will be used as pop-up menus should not be displayed in the main menu bar when the application is running. Hence those menus that you will be using as pop-up should be made invisible by unchecking the Visible property in the Menu Editor as shown in Figure 9.11.

#### Questions

1. Write code for a menu item by clicking on menu items in design mode.
2. Write code for Adding a Pop-up to a System Tray Icon.

### 9.8 Summary

- Menu editor is used to create new menu and bars.
- By adding access keys and shortcut keys we can improve keyboard access to menu commands.
- A separator bar is used to divide items into logical groups.
- A pop-up menu is a floating menu that is displayed over a form, independent of the menu bar.
- A modal dialog box must be closed before you can continue working with the rest of the application and modeless dialog boxes shift the focus between the dialog box and another form without having to close the dialog box.
- Predefined dialog box is the easiest way to add a dialog box into the application.

## 9.9 Keywords

**Access keys:** Allow the user to open a menu by pressing the ALT key and typing a designated letter.

**Context menus:** When the right mouse button was pressed; therefore, pop-up menus are also called context menus.

**Modal dialog box:** A window that appears on a computer screen, presenting information or requesting input.

**Modeless dialog box:** Modeless dialog box will allow the user to work with the parent window.

**Pop-up menu:** Pop-up menu is a floating menu that is displayed over a form, independent of the menu bar.



Lab Exercise

1. Create menu controls in the Menu Editor.
2. Write a step to assign an access key to a menu control in the Menu Editor.

## 9.10 Self Assessment Questions

1. The Execute Direct data-manipulation model is appropriate when
  - (a) You need to perform one-time-only operations on the data.
  - (b) You need to execute queries typed by users.
  - (c) You need to execute the same dynamic query several times during a single session of your application.
  - (d) You need to execute the same query over many sessions and from many different workstations.
2. The Menu Editor enables the programmer to create menu objects. These objects can have a variety of properties set and changed. Which property cannot be affected by using the Menu Editor?
  - (a) Checked
  - (b) Visible
  - (c) Picture
  - (d) Shortcut Key
  - (e) Enabled
3. In certain applications, objects provide a shortcut menu by using the right mouse button. This popup menu can be used to provide common commands for the selected object. Which item best describes the requirements to create a pop-up menu?
  - (a) Define the top-level and sub-level menu item, trap for the right mouse button in the MouseUp event, and call the form PopupMenu method.
  - (b) Define the menu items, trap for the right mouse button in the MouseUp event, and call only the selected menu items.

- (c) Use the pop-up property of the object.
  - (d) Assign a top-level menu to the pop-up menu property of the object.
  - (e) Use the Menu Editor, define the top-level menu name as "pop-up," and in the MouseUp event call the menu.
4. Assume that the Menu Editor has already been used to create a menu structure. For the menus to provide the functionality required, how are the menu items assigned program code?
- (a) By using the ItemCode property in the editor
  - (b) By using the Menu property in the Properties window
  - (c) By using the ItemCode property in the Properties window
  - (d) By selecting the menu item from the form and entering code into the opened Code window
  - (e) None of these
5. When a menu control array is created, runtime menu items can be added to the array. When the element's properties are set, where in the menu structure do the new items appear?
- (a) In the order specified by the NegotiatePosition property
  - (b) In the order specified by the array element
  - (c) Immediately below the preceding array element
  - (d) At the bottom of the menu specified at design time
  - (e) None of these
6. You can specify that a menu control will be a separator bar by
- (a) Supplying a space(" ") as the Name property
  - (b) Supplying a dash ("—") as the Caption property
  - (c) Supplying a dash ("—") as the Name property
  - (d) Supplying an underscore ("\_") as the Caption Property
7. Menus in Visual Basic can have their appearance changed to reflect the state of the application. To change menu items in VB at runtime, which methods can be used?
- (a) The Menu Editor
  - (b) Program code
  - (c) Program code and the Menu Editor
  - (d) Menu property of a form
  - (e) Negotiate property of a form
8. Command used to create new menus and menu bars/
- (a) Menu Editor
  - (b) Text editor
  - (c) Box editor
  - (d) None of the above

9. In the Menu Editor type of box used
- |                   |                       |
|-------------------|-----------------------|
| (a) Square box    | (b) List box          |
| (c) Rectangle box | (d) None of the above |
10. Pop-up menu displayed on
- |                       |                      |
|-----------------------|----------------------|
| (a) Task bar          | (b) Menu bar         |
| (c) None of the above | (d) All of the above |

### **9.11 Review Questions**

1. Need of zero order method, no of controls in form, Property used to add a menus at runtime, Property used to count number of items in a combobox,resize a label control according to your caption.
2. Which property of menu cannot be set at run time?
3. How do I make a menu popup from a Command Button?
4. How to create menus at run time in VB?
5. Visual Basic supports pop-up menus, those context-sensitive menus that most commercial applications show when you right-click on an user interface object. How pop-up menu can be created in Visual Basic? Explain with example.
6. Visual Basic forms can be used to host a menu bar. What two methods can be utilized by Visual Basic to create these menus?
7. When designing application menu bars, a variety of terminologies are used to refer to the different levels and positions held by a menu item. What is the name of a menu item that appears directly under a window's title bar?
8. When using the Menu Editor, how can the programmer define which menus will be at the top, which will be sub-items, and how the order is controlled?
9. Pop-up menus are displayed when the user selects the right mouse button. This displays a menu with options specific to the area that the user has selected. In which event, in Visual Basic, would a programmer trap for the use of the right mouse button?
10. Right-clicking on certain objects provides the user with a menu. What names are used to refer to this menu?
11. Menus can be created dynamically in Visual Basic by declaring a menu array and then loading a new element into the menu array. Once loaded, the new menu can have its properties set True or False. Why?
12. What is the use of MsgBox?
13. What is the use of Menu Editor in new menus and menu bars?
14. How to create a separator bar in the Menu Editor?

### Answers for Self Assessment Questions

- |                              |                  |                 |                 |
|------------------------------|------------------|-----------------|-----------------|
| 1. ( <i>a</i> and <i>b</i> ) | 2. ( <i>c</i> )  | 3. ( <i>a</i> ) | 4. ( <i>d</i> ) |
| 5. ( <i>b</i> and <i>c</i> ) | 6. ( <i>b</i> )  | 7. ( <i>b</i> ) | 8. ( <i>a</i> ) |
| 9. ( <i>b</i> )              | 10. ( <i>c</i> ) |                 |                 |

### 9.12 Further Readings



*Books*

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net."



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 10: Database Fundamentals and Connectivity Option**

### **CONTENTS**

Objectives

Introduction

10.1 The Data Control

10.2 The Data-Bound Controls

10.3 Creating Database Using Visual Database Manager

10.3.1 To Create a Database

10.3.2 Create a Table Using Visual DBA

10.3.3 To Create a Base Table

10.3.4 Populate a Table Using the Import Assistant

10.3.5 To Import Data From a File Using the Import Assistant

10.4 Connectivity of VB with Microsoft Access

10.4.1 Setting Up an ODBC Data Source

10.4.2 Sample Applications 2 and 3: Using ADO Code

10.5 SQL Server (Structured Query Language)

10.5.1 Overview

10.5.2 Getting Started with SQL Server 2005 Express

10.5.3 Connecting from Visual Basic 6

10.5.4 Multiple Recordsets, One Connection

10.5.5 Varchar (MAX)

10.5.6 Common Table Expressions

10.5.7 Common Language Runtime Integration

10.6 Summary

10.7 Keywords

10.8 Self Assessment Questions

10.9 Review Questions

10.10 Further Readings

### **Objectives**

*After studying this unit, you will be able to:*

- Understanding the data control and data bound controls
- Discuss creation of database using visual database manager
- Explain the connectivity of VB with microsoft access
- Discuss the SQL and its overview

## Introduction

The management of databases is the number one use of computers today. Airlines use databases to handle nearly 1.5 billion passenger reservations per year. The 6,500 hospitals in the United States utilize databases to document the care of over 30 million patients per year. Banks in the United States utilize databases to monitor the use of 350 million credit cards. Although databases vary considerably in size and complexity, most of them adhere to the fundamental principles of design discussed in this chapter. That is, they are composed of a collection of interrelated tables.

### 10.1 The Data Control

The data control enables to move around in a database from record to record and to display and manipulate data from the records in bound controls. This control displays a set of arrow buttons the user can manipulate to move through a database, and the records from that database are displayed in bound controls. We can see a data control operating with bound controls in Figure 10.1, where we placed our students table into a database and opened it with a data control.

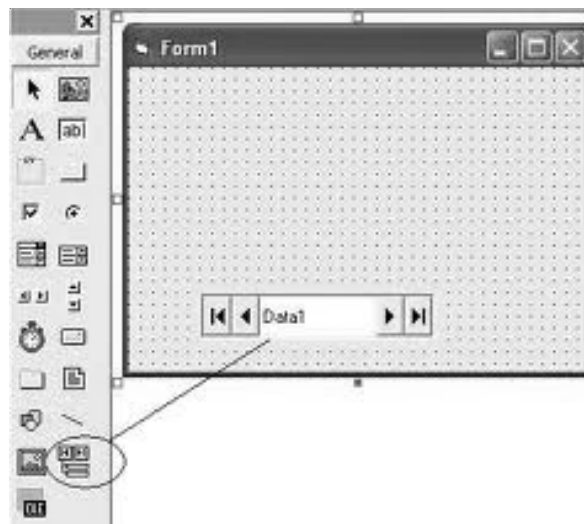


Figure 10.1: Using a Data Control.

In fact, we can perform most data access operations using the data control—without writing any code. Data-bound controls automatically display data from one or more fields for the current record, and the data control performs all operations on the current record. If the data control is made to move to a different record, all bound controls automatically pass any changes to the data control to be saved in the database. The data control then moves to the requested record and passes back data from the current record to the bound controls where it's displayed. When an application begins, Visual Basic uses data control properties to open a selected database, create a DAO Database object, and create a Recordset object. The data control's Database and Recordset properties refer to those **Database** and **Recordset** objects, and we can manipulate the data using those properties. For example, if we have an SQL statement to execute, we place that statement in the data control's **RecordSource** property, and the result appears in the **Recordset** property.

## **10.2 The Data-Bound Controls**

We can bind certain controls to the data control, the remote data control, and the ADO data control and those controls are called bound controls. To bind a control to a database control, we use properties like **DataSource** to specify the database control, and then use properties like **DataField** or **BoundColumn** to specify what field to display in the bound control, Here are the controls that can function as bound controls:

- Picture boxes
- Labels
- Text boxes
- Checkboxes
- Image controls
- OLE controls
- List boxes
- Masked edit controls
- Rich text boxes
- Combo boxes

In addition, there are special controls that are designed to be used as bound controls:

- DBList
- DBCombo
- FlexGrid
- MSFlexGrid

Finally, a number of bound controls are specially built to be used with the ADO control only:

- DataList
- DataCombo
- DataGrid

We'll see these controls at work in this chapter. That's it, then, for the overview of databases. We've seen how the process works in overview; now it's time to turn to the immediate solutions.

## **10.3 Creating Database Using Visual Database Manager**

To create a database using Visual DBA, use the Create Database dialog.

### **10.3.1 To Create a Database**

1. Select the Databases object category branch in the Database Object Manager window, and

then click the Add Object toolbar button .

The Create Database dialog appears.

The fields in this sample dialog are filled in as if creating the demodb database.



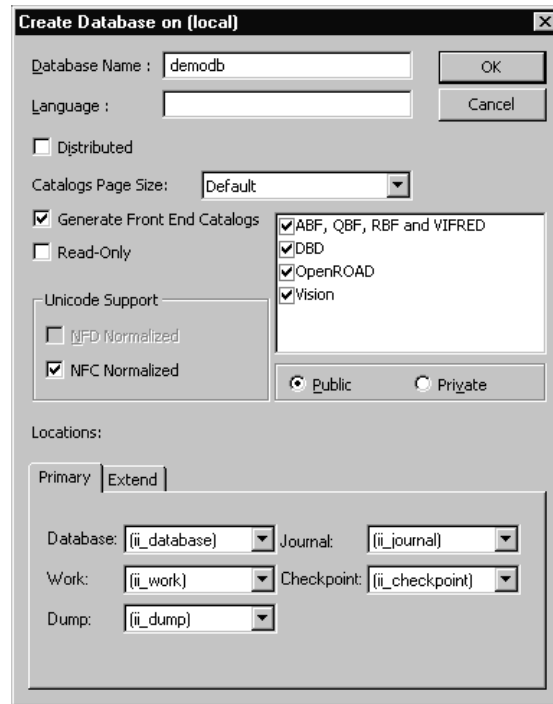


Figure 10.2: Creating Data Base.


2. Fill in the necessary information (for details, see the online help), and click OK. We are notified that the database has been created successfully.

### 10.3.2 Create a Table Using Visual DBA

After create a database, we can create its tables, indexes, and other database objects.

#### 10.3.3 To Create a Base Table

1. Expand the Databases object category branch in the Database Object Manager window, and then expand the branch of the database in which the table will be created.

2. Select the Tables object category branch and click the Add Object toolbar button .

The Create Table dialog appears.

3. Enter a unique, valid name for the table in the Table Name field. In the sample dialog below, the airport table in the demodb database is being created.



Notes

We can create a table from another table by selecting the Create Table as Select check box. For detailed information about each option, see the Visual DBA online help.

4. Enter a column name in the Name edit control in the column layout box (for example, **ap\_id**) and click Add. A column is added to the column layout box.

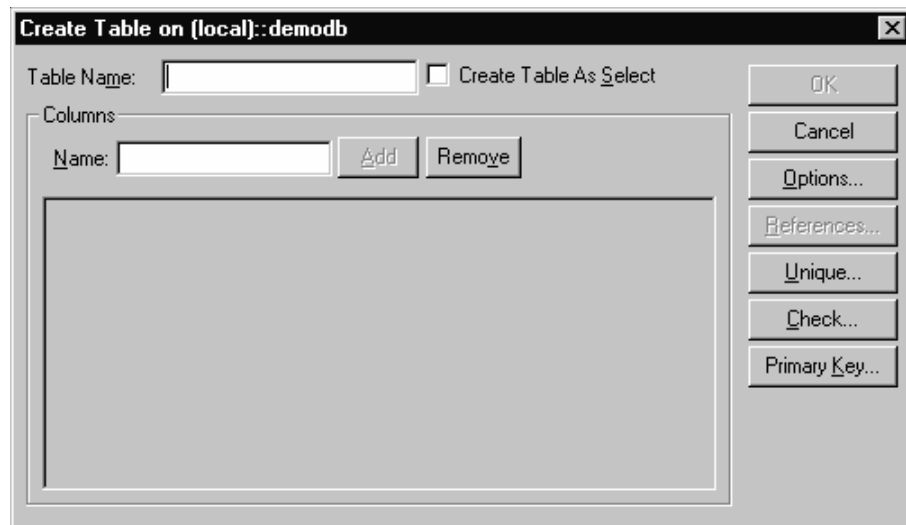


Figure 10.3: Creating Table.

5. Define the specifications of the column, such as data type, length, and primary key.



Example:

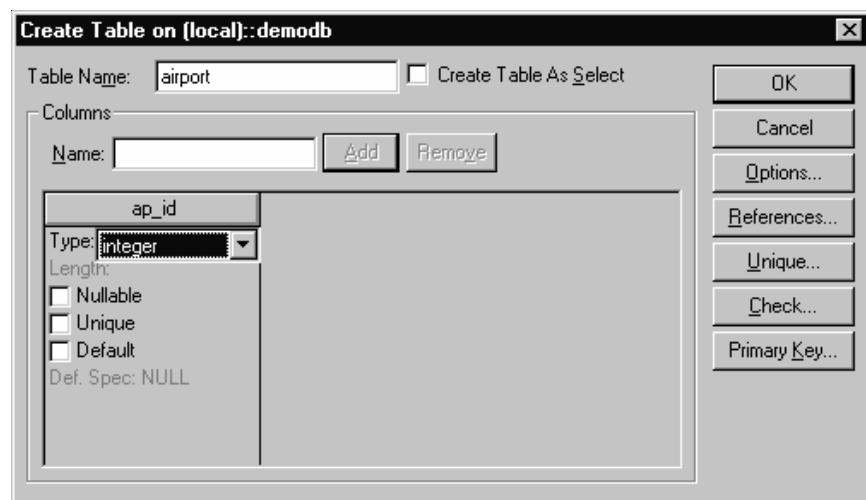


Figure 10.4: Entering Table Name.

6. Repeat steps 4 and 5 for each column we want to add to the table (for example, add an ap\_name column), and click OK. The table is created and appears under the Tables object category branch, as shown in this example:

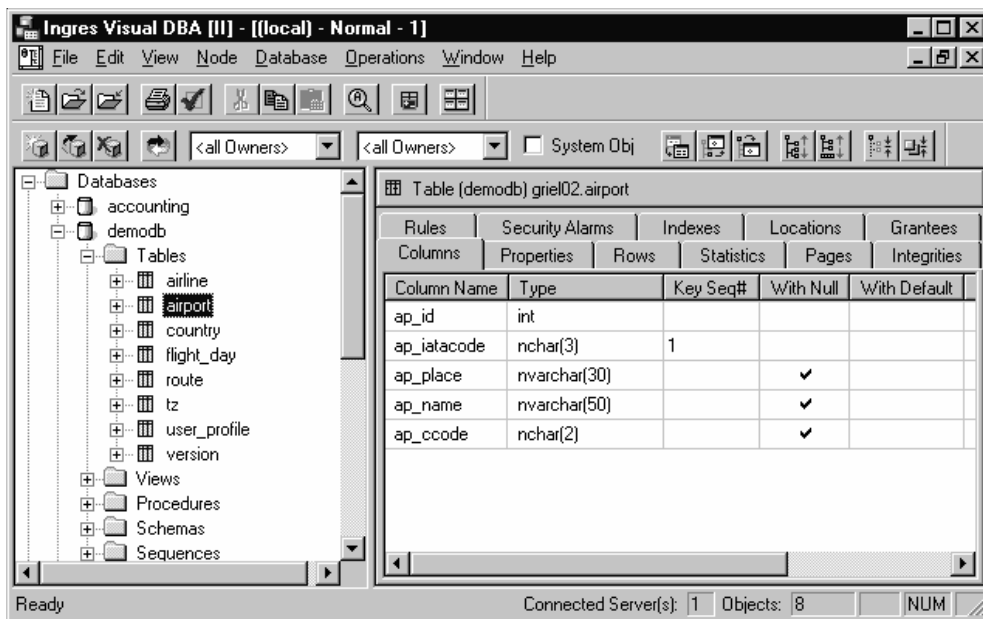


Figure 10.5: Object Category Branch.

After a table and its columns are defined, we can then populate the table with data from a file. We can also create an index to improve query processing and define a view, or virtual table, to limit access to specific columns.

### 10.3.4 Populate a Table Using the Import Assistant

We can populate a table with data by importing the data from a file. The Ingres Import Assistant can import data from various file types.


In this example, we use a CSV file located in %II\_SYSTEM%\ingres\demo\data\airport.csv.



Notes

The airport table in the demodb is already populated. This sample procedure creates a new table called **temp**, which we can delete (drop) after this exercise.

### 10.3.5 To Import Data From a File Using the Import Assistant

1. Click the Ingres Import Assistant button  on the toolbar in Visual Manager. Step 1 of 4 of the Import Assistant is displayed.
2. Use the browse button in the File to be imported control to locate the file.
3. Expand the (local) node in the Into Ingres Table control, and then expand the demodb database. Select the <new table> option and enter **temp** into the Ingress Table to be Created field. Press Next. Step 2 of 3 of the Import Assistant (File Format Identification) is displayed. The Import Assistant displays each of its interpretations of the data as a tab.

4. Select the Csv1 tab, which shows that there are five columns of data. Check the Tab corresponds to expected results box and press Next. Step 3 of 4 of the Import Assistant (Column Properties) is displayed. The utility assigns default values to column properties.
5. Double-click the "with null" field in the first column and select "not null" from the dropdown list.

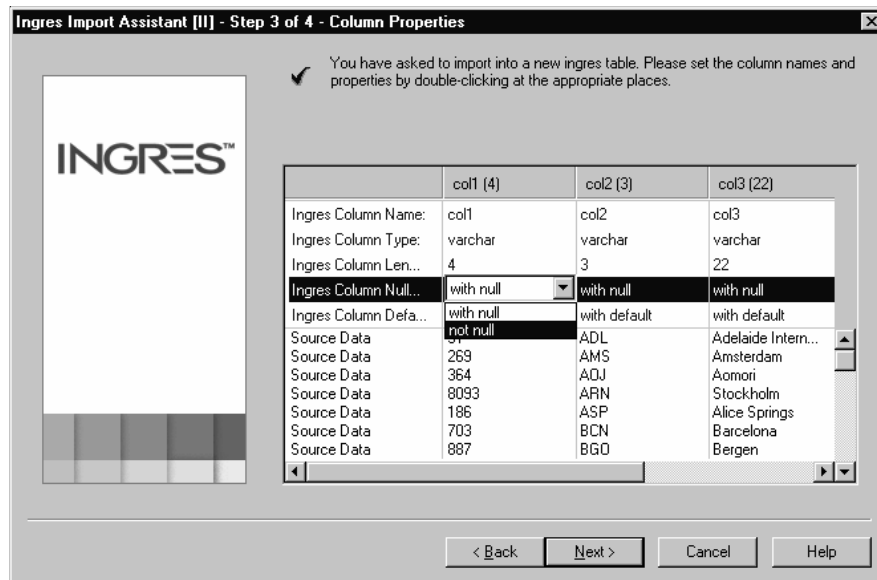


Figure 10.6: Ingress Import Assistant.



Notes

When importing into Unicode columns, use the data type char or varchar. Ingres will coerce the values to Unicode using the appropriate Unicode mapping, indicated by the value of the Ingres environment variable II\_CHARSET.

Press Next. Step 4 of 4 of the Import Assistant (Import Preview) is displayed.



Notes

If the file was large, we could commit every 1000 rows to avoid filling the transaction log file.

6. Press the Finish Button The new table is populated. We are asked if we want to import another file.
7. Press No.The Import Assistant closes.

## 10.4 Connectivity of VB with Microsoft Access

### 10.4.1 Setting Up an ODBC Data Source

Follow the steps below to set up an ODBC Data Source (this process is also called "setting up a DSN", where "DSN" stands for "Data Source Name"). These steps assume Windows 2000 for the operating system. On other versions of Windows, some steps may vary slightly.

- Via Windows, **Control Panel**, double-click on **Administrative Tools**, then **Data Sources (ODBC)**. The **ODBC Data Source Administrator** screen is displayed, as shown below. Click on the **System DSN** tab.

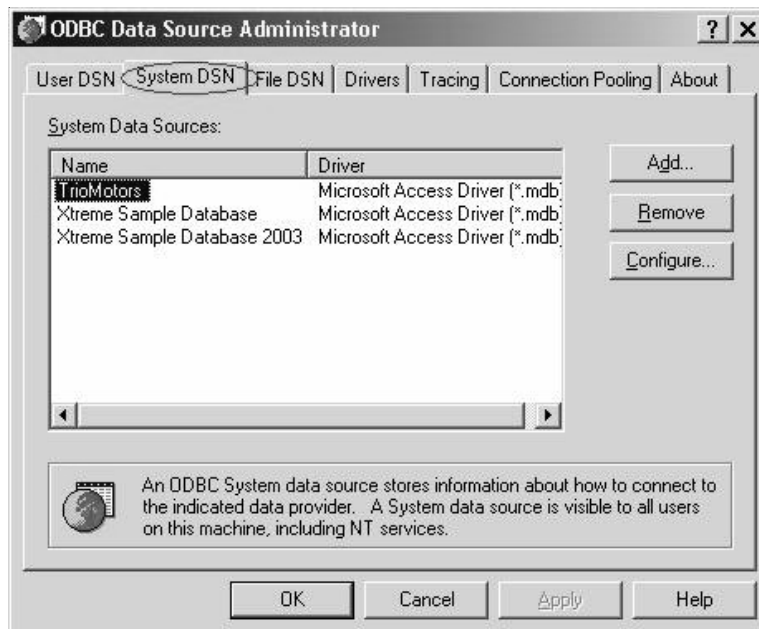


Figure 10.7: Data Source Administrator.

- Click the **Add** button. The **Create New Data Source** dialog box will appear. Select **Microsoft Access Driver (\*.mdb)** from the list and click the **Finish** button.

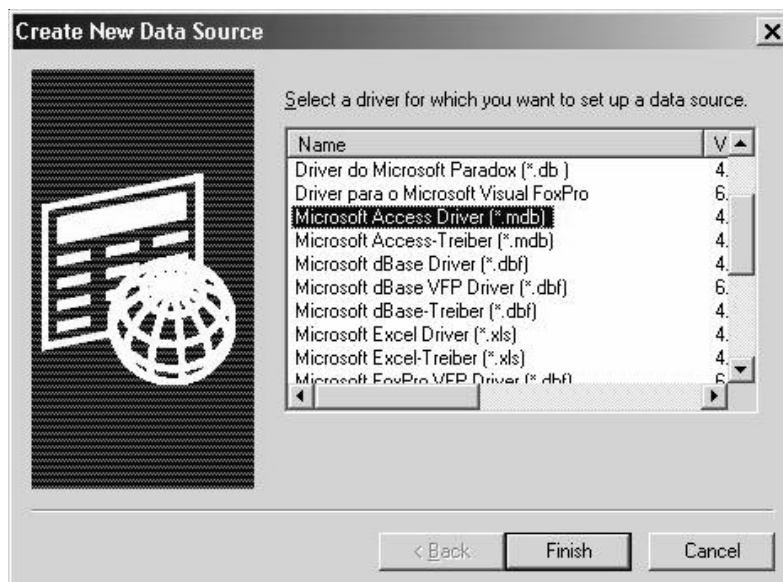


Figure 10.8: Select Data Source.

- The **ODBC Microsoft Access Setup** dialog box will appear. For **Data Source Name**, type **Biblio**. If desired, we can type an entry for **Description**, but this is not required.

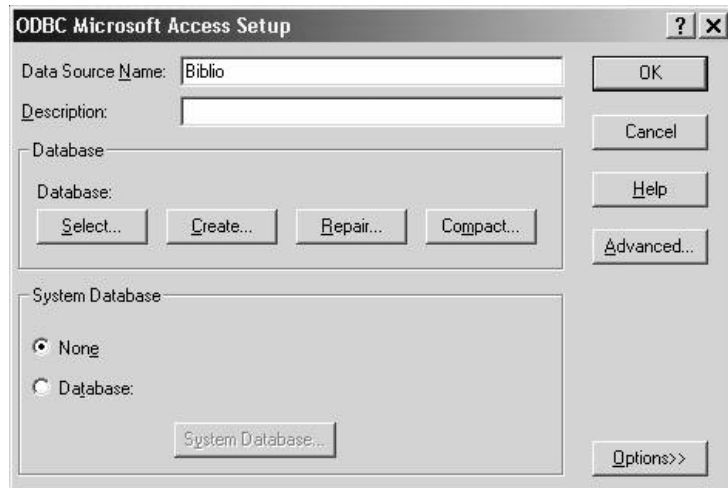


Figure 10.9: MS Access Setup.

- Click the **Select** button. The **Select Database** dialog box appears. On a default installation of VB6 or Visual Studio 6, the **BIBLIO.MDB** sample database should reside in the folder **C:\Program Files\Microsoft Visual Studio\VB98**. Navigate to that folder, select **BIBLIO.MDB** from the file list, and click **OK**.

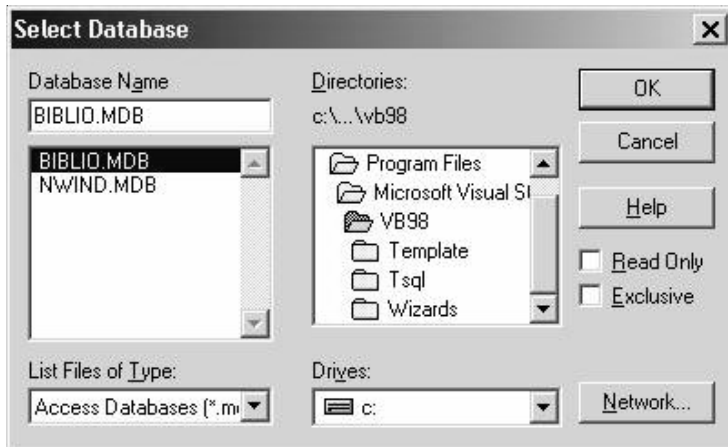


Figure 10.10: Data Base Selection.



Notes

If VB was installed in a different location on system, navigate to the appropriate folder. If we do not have the **BIBLIO.MDB** sample database file on system at all, we can download it here. In that case, copy the file to the folder of choice, and navigate to that folder to select the database for this step.

- When we are returned to the ODBC Microsoft Access Setup screen, the database we selected should be reflected as shown below. Click **OK** to dismiss this screen.

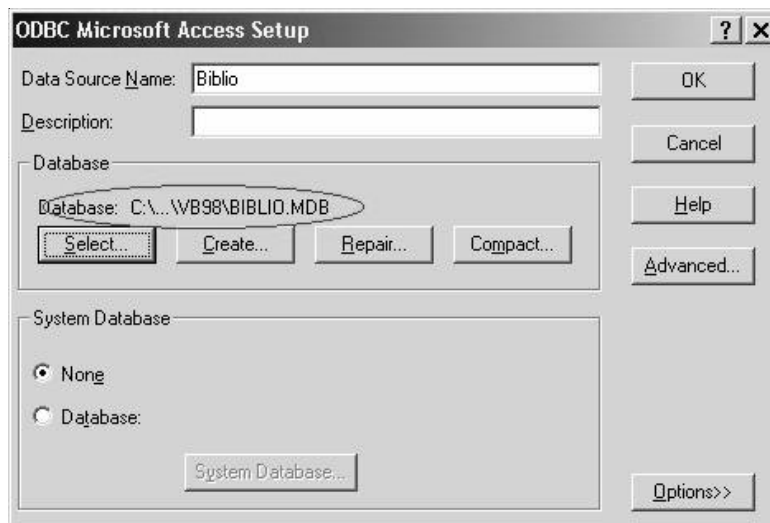


Figure 10.11: Path of Data Base.

- When we are returned to the ODBC Data Source Administrator screen, the new DSN should appear as shown below. Click OK to dismiss this screen.

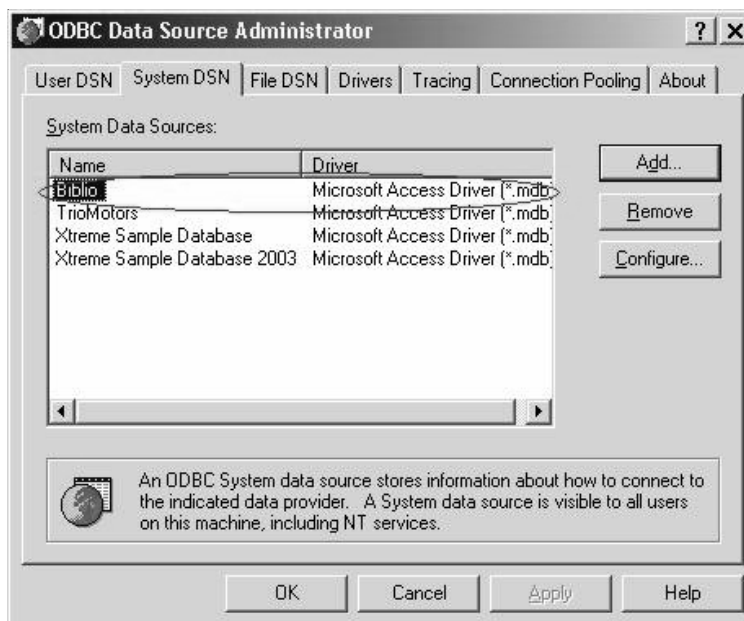


Figure 10.12: Data Source Administrator.

At this point, the Biblio database is ready to be used with RDO in the sample application.

#### Sample Application 1: Using the ADO Data Control (ADODC)

To build the first sample application, follow the steps below.

- Start a new VB project, and from the **Components** dialog box (invoked from the **Project** -> **Components** menu), select **Microsoft ADO Data Control 6.0 (SPx)** as shown below and click **OK**.

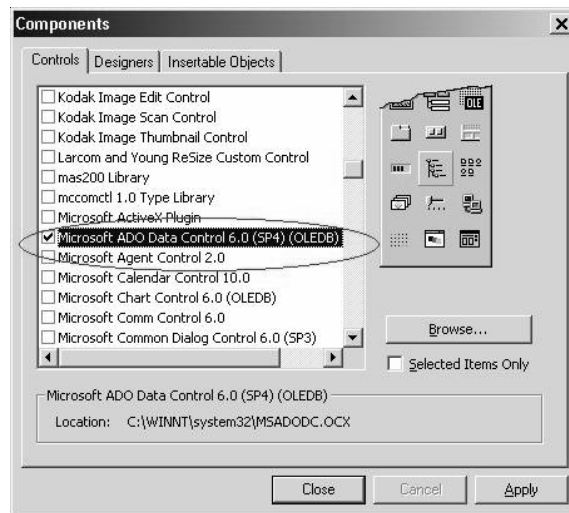


Figure 10.13: Components.

The ADO Data Control should appear in toolbox as shown below:



Figure 10.14: Tool Box.



- Put an ADO Data Control on form, and set the properties as follows:

| Property       | Value                 |
|----------------|-----------------------|
| Name           | adoBiblio             |
| DataSourceName | Biblio                |
| SQL            | select * from authors |

- Now put three text boxes on the form, and set their Name, DataSource, and DataField properties as follows:

| Name        | DataSource | DataField |
|-------------|------------|-----------|
| txtAuthor   | adoBiblio  | Author    |
| txtAuID     | adoBiblio  | Au_ID     |
| txtYearBorn | adoBiblio  | Year Born |

- Save and run the program. Notice how it works just like the other data control.
- Now change the SQL property of the data control to **select \* from authors order by author** and run the program again. Notice the difference.
- Change the SQL property back to what it was and add three command buttons to the form, and set their Name and Caption properties as follows:

| Name         | Caption               |
|--------------|-----------------------|
| cmdNameOrder | Order by <u>N</u> ame |
| cmdYearOrder | Order by <u>Y</u> ear |
| cmdIDOrder   | Order by <u>I</u> D   |

- Put the following code in the **cmdNameOrder\_Click** event: `adoBiblio.SQL = "select * from authors order by author" adoBiblio.Refresh`
- Put the following code in the **cmdYearOrder\_Click** event: `adoBiblio.SQL = "select * from authors order by [year born]" adoBiblio.Refresh`
- Put the following code in the **cmdIDOrder\_Click** event: `adoBiblio.SQL = "select * from authors order by au_id" adoBiblio.Refresh`
- Save and run the program and see what happens when we click the buttons.

A screen-shot of the sample app at run-time is shown as follows:

#### 10.4.2 Sample Applications 2 and 3: Using ADO Code

The Property database contains just one table called “Property”. The columns of this table are defined as follows:

| Column Name  | Data Type             | Notes                                                                                                                                                                                                                  |
|--------------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PROPNO       | Number (Long Integer) | A number that uniquely identifies the property in the table. Should be treated as the Primary Key (although it is not defined as such in the sample database).                                                         |
| EMPNO        | Number (Long Integer) | A number that identifies the real estate agent selling the property. In a real system, this would be the foreign key to the employee number in an Employee table (such a table is not present in the sample database). |
| ADDRESS      | Text (20)             | The street address of the property.                                                                                                                                                                                    |
| CITY         | Text (15)             | The city where the property is located.                                                                                                                                                                                |
| STATE        | Text (2)              | The state where the property is located (2-characterUS state abbreviation).                                                                                                                                            |
| ZIP          | Text (5)              | The zip code where the property is located.                                                                                                                                                                            |
| NEIGHBORHOOD | Text (15)             | The descriptive name of the neighborhood in which the property is located.                                                                                                                                             |

|           |                       |                                                                                                                                                                                                      |
|-----------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HOME_AGE  | Number (Long Integer) | Age in years of the home. (A better table design choice would be to have this field be the date in which the property was built and have the application compute the age based on the current date.) |
| BEDS      | Number (Long Integer) | Number of bedrooms in the property.                                                                                                                                                                  |
| BATHS     | Number (Single)       | Number of bathrooms in the property (allows for a decimal value such as 2.5, indicating 2 ½ bathrooms – i.e., 2 full bathrooms and 1 “powder room”).                                                 |
| FOOTAGE   | Number (Long Integer) | The footage of the property.                                                                                                                                                                         |
| ASKING    | Number (Long Integer) | Asking price of the property in whole dollars.                                                                                                                                                       |
| BID       | Number (Long Integer) | Bid amount of the potential buyer in whole dollars.                                                                                                                                                  |
| SALEPRICE | Number (Long Integer) | Sale price (amount the property actually sold for) in whole dollars.                                                                                                                                 |

Before coding or running sample application 2 or 3, we must set up an ODBC data source as was done for the previous sample application. After downloading the file, move it to the folder of choice. Then follow the exact same steps as before to set up the DSN, with these two exceptions:

- (1) On the **ODBC Microsoft Access Setup** dialog box, type **PropDB** for the **Data Source Name**.
- (2) In the **Select Database** dialog box, navigate to the location where we have placed the **PROPERTY.MDB** file.

## 10.5 SQL Server (Structured Query Language)

### 10.5.1 Overview

Visual Basic developers have long used Access databases as a solution for local storage of application data. Access made it very simple to ship a database with application, and connecting to the database was no more complicated than putting the path to the database in the connection string. For more complex scenarios, developers could scale up to the Microsoft Desktop Storage Engine (MSDE). This provided developers with SQL Server functionality and reliability, but at the expense of ease of use. With MSDE, we had to attach databases before they could be connected to. When the application shut down, the database file was still in use by the MSDE engine, making uninstall and reinstall more complicated. MSDE also did not provide any administrative interface, so any administration of the database had to be done through command line tools. In addition, MSDE was difficult to deploy with an application.

As part of the effort to develop Visual Studio 2005 and SQL Server 2005, Microsoft is addressing all these limitations of MDSE, replacing it with a new product called SQL Server 2005 Express.

While information about SQL Server 2005 Express has so far been targeted at Framework developers, this article will show how SQL Server 2005 Express is a boon for Visual Basic 6 development as well. SQL Server 2005 Express will provide the developer power of SQL Server, but also the ease of use of access, by letting we connect directly to a database file on disk. Best of all, it's completely free, and can be redistributed with application.

### 10.5.2 Getting Started with SQL Server 2005 Express

If we want to work with SQL Server 2005 Express, the first step is to download and install it. SQL Server 2005 Express is part of the family of Express products that can be found at <http://lab.msdn.microsoft.com/express/>. Prior to installing SQL Server 2005 Express,

Unlike MSDE, SQL Express also includes an administrative tool known as Express Manager, which lets we create databases and execute queries (see Figure 10.15).

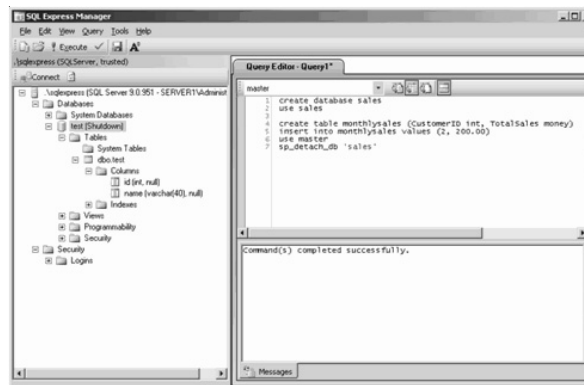


Figure 10.15: Using Express Manager to Create a Sample Database.

### 10.5.3 Connecting from Visual Basic 6

To start working with Express, a simple user interface was created that uses an MSHFlexGrid to display data (see Figure 10.16).

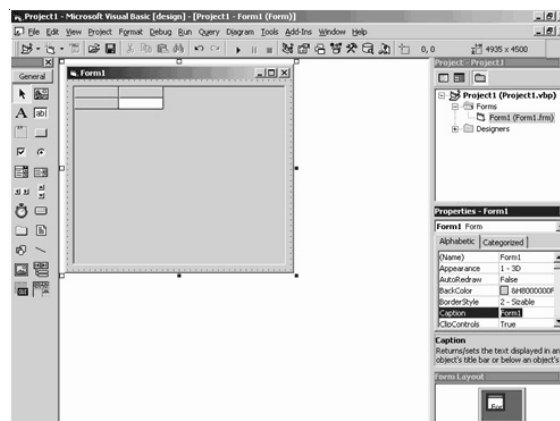


Figure 10.16: Using Grids to Display Data.

Then, a few simple lines of code (see Listing 1) can be written that populate the grid from a SQL Server database, which is connected to by specifying the path to its .mdf file.

### Listing 1. Using Visual Basic 6 to retrieve information from SQL Server 2005

#### Express 2005 by file path

```
Dim cn As ADODB.Connection
Set cn = New Connection
cn.ConnectionString = "Provider=SQLNCLI.1;Integrated Security=SSPI;"
& _
 "Persist Security Info=False;" & _
 "AttachDBFileName=" & App.Path & "\northwnd.mdf;Data
Source=server1\squlexpress"
cn.Open

Dim rs As Recordset
Set rs = New Recordset
rs.Open "Select * from orders", cn
Set MSHFlexGrid1.DataSource = rs
```

This is revolutionary functionality. This means that we can simply ship SQL Server database as an .mdf file with application. We don't need to do anything to install the database. we just put the path to the database in the connection string, and we can select, insert, update, delete, call stored procedures, and access any other database functionality. This is also great functionality for Web applications, which can now just keep a copy of the .mdf file in the Web application directory, and use it as easily as an Access database.

In Listing 1, we can see that the path to the database is specified through the **AttachDBFileName** value in the connection string. We'll also notice that this connection string is not using the typical SQL OLE-DB provider. When we install SQL Server 2005 Express, it installs a new OLE-DB provider, called the SQL Native Client, that gives we access to some of the new SQL Server 2005 functionality—specifically, the ability to attach to a database by file name. This is specified in the connection string with the statement **Provider=SQLNCLI**. Once the connection string is specified, we can retrieve recordsets, perform updates, and do any other database operations, exactly as we would normally do with SQL Server.



Task

Get Started with SQL Server 2005 Express.

#### 10.5.4 Multiple Recordsets, One Connection

Another key feature of SQL Server 2005, and SQL Server 2005 Express, is the ability to have multiple result sets off of a single connection. We are probably familiar with the situation in which we are looping through a recordset, and, based on the information in the recordset, we need to open another connection to the database and retrieve additional information. Because of key enhancements made to SQL Server, we no longer need to open multiple connections to the same database. Multiple Active Result Sets (MARS) lets we work with multiple recordsets through a single connection. Consider the code in Listing 2.

**Listing 2. Using multiple active recordsets from Visual Basic 6**

```
\
\ Connect to the database
\
Dim cn As ADODB.Connection
Set cn = New Connection
cn.ConnectionString = "Provider=SQLNCLI.1;Integrated Security=SSPI;" & _
 "Persist Security Info=False;" & _
 "AttachDBFileName=" & App.Path & "\northwnd.mdf;Data
Source=server1\sqlexpress"
cn.Open

\
\ Get all the orders
\
Dim rsOrders As Recordset
Set rsOrders = New Recordset
rsOrders.Open "Select * from orders", cn
Do While Not rsOrders.EOF
 \
 \ If the order matches some custom business logic then get
the details for
 \ that order, without opening a new connection.
 \
 If SomeBusinessLogic(rsOrders("CustomerID")) Then
 Dim rsDetails As Recordset
 Set rsDetails = New Recordset
 \
 \ Open a new recordset using the same connection.
Normally it's not
 \ possible to have two recordsets simultaneously using
the same
 \ connection, but MARS makes this possible
 \
 rsDetails.Open "Select sum(quantity * unitprice) as total
" & _
 "from [order details] " & _
 "where OrderID=" & rsOrders("OrderID"), _
 cn
 grandTotal = grandTotal + rsDetails("total")
 End If
```

```
rsOrders.MoveNext
Loop
lblTotalOrders = grandTotal
```

In this case, a recordset is retrieved that contains all the orders (**rsOrders**). Based on some custom business logic, the details for certain orders need to be retrieved. We can see that an additional recordset known as **rsDetails** is opened to retrieve the details, using the same connection as **rsOrders**, while **rsOrders** is still open and being used. In the past, this wouldn't have been possible. This allows us to create a single global connection object for application, and to use this throughout application for all access to a given database.

### 10.5.5 Varchar (MAX)

In previous versions of SQL Server, we would use a **VARCHAR** column to store text. The problem was that a **VARCHAR** column was limited to 8000 characters. If we needed to store more than that, we had to use the **TEXT** type, and **TEXT** was significantly harder to work with. SQL Server 2005 (and SQL Server 2005 Express) supports a new data type known as **VARCHAR (MAX)**. This type can be treated exactly like a **VARCHAR**, and it can store 2 GB of data per row.

For binary data, SQL Server 2005 Express also includes a **VARBINARY (MAX)** type, which replaces the **IMAGE** type.

### 10.5.6 Common Table Expressions (CTE)

It's common, when working with data, to need to create intermediate results. These can show up in the form of temporary tables, or views. Often, this intermediate information is needed only while a given calculation is being done, and it can then be discarded. With SQL Server 2005 Express, a new mechanism for these kinds of scenarios is the common table expression (CTE). Simply put, a CTE lets us treat the results of a select statement as a table. To look at a concrete example, consider the CTE in Listing 3, which returns the region with the most territories.

#### Listing 3. Common table expression (CTE) with SQL Server 2005

```
WITH CountTerritories (RegionID, TerritoryCount) AS
 SELECT
 RegionID,
 count(*) AS TerritoryCount
 FROM Territories
 GROUP BY RegionID
)
SELECT * FROM CountTerritories WHERE TerritoryCount =
(
 SELECT max(TerritoryCount) FROM CountTerritories
)
```

Here, **Count Territories** is created as a sort of temporary view that's used just for this single select statement. Once created, it can be queried against in the subsequent **SELECT** statement, as though it were a table.

### 10.5.7 Common Language Runtime Integration

The details of this are beyond the scope of this article, but it's also worth noting that with SQL Server 2005, we can build DLLs that run inside of SQL Server. This lets we implement triggers, stored procedures, user-defined functions, and so, using languages such as Visual Basic or Visual C#.



Structured Query Language (usually referred to as SQL) is the language we use to specify the data included in a Record set. It can also provide we with the vehicle for specifying changes to data through the Database object's Execute method.



#### Case Study

#### Connecting to an Access Database Using the VB Data Control

##### Steps

1. Open a new Visual Basic project.
2. Put a data control (an intrinsic control, located in the VB toolbox) on the form and set the properties as follows:

| Property     | Value                               |
|--------------|-------------------------------------|
| (Name)       | datAuthors                          |
| Caption      | Use the arrows to view the data     |
| Connect      | Access ( <i>default</i> )           |
| DatabaseName | ..\biblio.mdb                       |
| DefaultType  | UseJet ( <i>default</i> )           |
| RecordSource | Authors ( <i>choose from list</i> ) |



#### Notes

When we use the Data Control in a project, the properties that must be set are DatabaseName and RecordSource, in that order. DatabaseName is the name of the database we want to use, and the RecordSource is the name of the table in that database that we want to use.

3. On form, create a text box for each field in the table, with labels. Set the properties of the three textboxes as follows:



| Name        | DataSource | DataField |
|-------------|------------|-----------|
| txtAuthID   | datAuthors | Au_ID     |
| txtAuthor   | datAuthors | Author    |
| txtYearBorn | datAuthors | Year Born |

In addition, set the **Enabled** property of txtAuthID to **False**.

When we want a control (such as a text box) to display data from a database, the properties that must be set are **DataSource** and **Datafield**. The **DataSource** is the name of the data control on the form (it should already be configured), and the **DataField** is the name of the particular field in the database that should be displayed in the control (this field will be in the table that was chosen for the RecordSource of the data control).

At this point, your form should resemble the screen-shot below:

4. Save and run the project. Use the arrows on the data control to scroll through the data.
5. On any record, change the data in the author name or year born field. Move ahead, then move back to the record you changed. Note that your changes remain in effect. The data control automatically updates a record when you move off of the record.

#### Questions

1. Connect to an Access Database Using Navigation Buttons with a Data Control.
2. Connect to an Access Database Using the BOFAction and EOFAction Properties of the Data Control.

## 10.6 Summary

- Structured query language is a standard programming language used for accessing and maintaining database.
- Data bound control, bind a control to a database properties like data source.
- Tables using import assistant in which we can populate table with data by importing data from the file.
- Varcher (MAX) is to use the TEXT type and TEXT type data for SQL server.
- Common table Expression (CTE) is a mechanism to result of a select statement as a table for SQL.

## 10.7 Keywords

**BoundColumn Class:** A column type for the DataGrid control that is bound to a field in a data source.

**Count Territories:** It is created as a sort of temporary view that's used just for this single select statement. Once created, it can be queried against in the subsequent SELECT: statement, as though it were a table.

**Data Control Language (DCL):** It is a computer language and a subset of SQL, used to control access to data in a database.

**DataSource:** It is data-provider-independent description of a set of objects that will be loaded, edited and saved within the user interface of application.

**MSHFlexGrid:** The Microsoft Hierarchical FlexGrid (MSHFlexGrid) control displays and operates on tabular data. It allows complete flexibility to sort, merge, and format tables containing strings and pictures. When bound to a data control, **MSHFlexGrid** displays read-only data.

**Recordset:** It is a data structure that consists of a group of database records, and can either come from a base table or as the result of a query to the table.

**Structured Query Language (SQL):** It is a standard programming language used for accessing and maintaining a database. The key feature of the SQL is an interactive approach for getting information from and updating a database.

**Varchar: Variable Character Field** is a set of character data of indeterminate length. The term varchar refers to a data type of a field (or column) in a database management system.



Lab Exercise

1. Create Database Using Visual Database Manager.
2. Connect SQL Server\_from Visual Basic 6.
3. Connect VB with Microsoft Access.

### **10.8 Self Assessment Questions**

1. After you have executed a SQL Server stored procedure that implements a return value, you can check the value that the stored procedure returned by
  - (a) Using the individual variables corresponding to the stored procedure's Output parameters in the Parameters array argument to the Command object's Execute method.
  - (b) Checking the final element of the Command object's Parameters collection.
  - (c) Checking element 0 of the Command object's Parameters collection.
  - (d) Checking the element of the Command object's Parameters collection that corresponds to the last Output parameter declared in the stored procedure.
2. A SQL Server stored procedure designed to return records to a VB app using ADO (Select all that apply.)
  - (a) Must have at least one Output parameter.
  - (b) Must have at least one Input parameter.
  - (c) Must implement a return value.
  - (d) Doesn't necessarily have any parameters.
3. A SQL statement to delete all records from the employees table would read
  - (a) DELETE "" FROM employees
  - (b) DELETE \* FROM employees
  - (c) DELETE FROM employees
  - (d) DELETE FROM employees WHERE \*
4. Which statements are true of the GetDataMember event?
  - (a) You initiate a data connection in its event procedure.
  - (b) You set its first parameter to the control's data connection.
  - (c) It fires whenever a bound control needs to refresh its data.
  - (d) You set its second parameter to the control's Record set.
5. VARCHAR column was limited to 8000 characters.
  - (a) True
  - (b) False
6. Setting Up an ODBC Data Source process called.
  - (a) "setting up a DSM"
  - (b) "setting up a DSN"
  - (c) "setting up a DSK"
  - (d) "setting up a DSA"
7. To bind a control to a database control, we use properties like **DataField** to specify the database control.
  - (a) True
  - (b) False

8. Give names of those controls which can function as bound controls.
  - (a) Labels
  - (b) Text boxes
  - (c) List boxes
  - (d) All of the above.
9. Path to the database is specified through the value in the connecting string
  - (a) Provider=SQLNCLIB.
  - (b) AttachDBFileName
  - (c) MSHFlexGrid
  - (d) All of the above.
10. SQL Express also includes an administrative tool known as Express Manager
  - (a) False
  - (b) True

### **10.9 Review Questions**

1. What is the default property of data control?
2. What are two validate with Data Control?
3. Name some advantages of stored procedures over inline SQL statements.
4. What keyword begins a SQL statement to retrieve records?
5. Give the name and brief on those controls which can function as bound controls.
6. Explain data control's Database and Recordset properties.
7. What is MSHFlexGrid? Explain with suitable examples.
8. Explain some useful TextBox Validations.
9. How do I use charts in Visual Basic?
10. How do I display pictures using a ListBox?
11. How do I add pictures to menu items?
12. How do I change one colour to another in a picture?
13. How do I resize MDI Child Forms?
14. How to bind two controls to the same DataTable without having changes in one control also change the other control?

### **Answers for Self Assessment Questions**

- |        |         |        |           |
|--------|---------|--------|-----------|
| 1. (c) | 2. (d)  | 3. (c) | 4. (a, d) |
| 5. (a) | 6. (b)  | 7. (b) | 8. (d)    |
| 9. (b) | 10. (b) |        |           |

### **10.10 Further Readings**



*Books*

G. Cornell , "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net."



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 11: Data Base Access**

### **CONTENTS**

|                                                    |
|----------------------------------------------------|
| Objectives                                         |
| Introduction                                       |
| 11.1 ADO Data Model                                |
| 11.2 Data Access Objects (DAO)                     |
| 11.3 The Remote Data Control                       |
| 11.3.1 Opening a Database with an ADO Data Control |
| 11.3.2 The ADO Data-Bound Controls                 |
| 11.4 Data Base Access Using the ADO Data Control   |
| 11.4.1 Setting Up the ADO Data Control             |
| 11.4.2 Creating an ADO Data Control                |
| 11.5 ADO and ADOCE                                 |
| 11.6 Summary                                       |
| 11.7 Keywords                                      |
| 11.8 Self Assessment Questions                     |
| 11.9 Review Questions                              |
| 11.10 Further Readings                             |

### **Objectives**

*After studying this unit, you will be able to:*

- Discuss ADO data model
- Explain Data access object (DAO)
- Discuss the remote data control
- Explain data base access using the ADO data control
- Explain ADO and ADOCE

### **Introduction**

ADO provides a new and powerful means of accessing data stored in databases. As a single, high-level technology for accessing all types of databases, ADO is simpler and more efficient to use than the two older data access technologies, DAO (Data Access Objects) and RDO (Remote Data Objects). ADO combines the functionality of DAO and RDO into one interface and provides even more features. Rather than having to choose DAO or RDO depending on the type of database you need to access, you can use ADO to access any type of database or file. For example, while DAO can connect to Microsoft Access .MDB databases, ADO can access HTML, plain text, Internet, spreadsheet, and e-mail files, as well as .MDB and other types of databases.

### **11.1 The ADO Data Model**

ADO is a language-neutral object model that lets you manipulate data accessed by an underlying OLE DB provider. (An OLE DB provider is a data manager that interfaces directly with a database. For example, Microsoft Jet is the data manager on which Microsoft Access is built.) ADO's object model is simpler than DAO's, containing fewer objects and more properties, methods, and events, and combining much of DAO's functionality into single objects. Many of the objects in the ADO data model are similar to RDO objects, but with enhanced functionality. For example, ADO's Recordset object contains records returned from a database plus the cursor for those records. (A cursor helps manage and manipulate a set of data.) The Recordset object is similar to RDO's rdo Resultset object. With ADO, however, you can open a Recordset object without explicitly connecting to the database source. And if you do connect to the database, you can open multiple Recordset objects on the same connection. You can learn more about the ADO data model from the references listed in where can I find out more about ADO?

### **11.2 Data Access Objects (DAO)**

When Visual Basic first started working with databases, it used the Microsoft Jet database engine, which is what Microsoft Access uses. Using the Jet engine represented a considerable advance for Visual Basic, because now you could work with all kinds of data formats in the fields of a database: text, numbers, integers, longs, singles, doubles, dates, binary values, OLE objects, currency values, Boolean values, and even memo objects (up to 1.2GB of text). The Jet engine also supports SQL, which database programmers found attractive. To support the Jet database engine, Microsoft added the data control to Visual Basic, and you can use that control to open Jet database (.mdb) files. Microsoft also added a set of Data Access Objects (DAO) to Visual Basic:

- DBEngine—The Jet database engine
- Workspace—An area can hold one or more databases
- Database—A collection of tables
- TableDef—The definition of a table
- QueryDef—The definition of a query
- Recordset—The set of records that make up the result of a query
- Field—A column in a table
- Index—An ordered list of records
- Relation—Stored information about the specific relationship between tables

### **11.3 The Remote Data Control**

Like the data control, the remote data control gives you access to a database and displays data in bound controls. Unlike the data control, however, you use the remote data control to access ODBC data sources (which can include databases built with all the popular commercial database programs).

As with the data control, if the remote data control is instructed to move to a different row, all bound controls automatically pass any changes to the remote data control to be saved to the ODBC data source. The remote data control then moves to the requested row and passes back data from the current row to the bound controls where it's displayed.

In fact, the remote data control behaves like the data control in most respects, with some differences; for example, you can treat the remote data control's **SQL** property like the data control's.



*Did u know?*

DAO Performs Better Than ADO When Accessing Jet Databases When accessing Jet databases (that is, MS Access), DAO will perform better than ADO, because DAO was optimized for Jet.

### 11.3.1 Opening a Database with an ADO Data Control

To add a new ADO data control to a form, follow these steps:

1. Select the Project\Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Microsoft ADO Data Control entry in the Controls list box.
4. Click on OK to close the Components dialog box.
5. This adds the ADO data control tool to the toolbox; draw that control as you want it on your form.
6. Connect the ADO data control's Connection object to a data source with the **ConnectionString** property, separating items in that string with semicolons. At the least, you should specify the **Provider** (the type of OLE DB) and **Data Source** (database name) values in the **ConnectionString**.

Let's see an example. Here, we'll connect an ADO data control to the database we've constructed in the early parts of this chapter, db.mdb. To do that, add an ADO data control, **Adodc1**, to a form, and set its **ConnectionString** property to specify the data provider type and the data source for that database like this:

```
"PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=c:\vbbb\ado\db.mdb;"
```



*Caution*

One way of connecting an ADO control to a database easily is with the Data Form Wizard, which generates the connection string for you automatically.

Next, set the ADO data control's **RecordSource** property to the table to work with, which is students in our example database, db.mdb.

Now you've connected a database to the ADO data control. To connect the ADO data control to bound controls.

### 11.3.2 The ADO Data-Bound Controls

There are three data-bound controls that are specially optimized for use with the ADO data control: DataGrid controls, DataCombo controls, and DataList controls (don't confuse these controls with the non-ADO optimized data-bound controls like the DBCombo and DBList controls). These controls are specifically designed to work with ADO data controls and won't work with standard controls like the data control.

To add these controls to a program, follow these steps:

1. Select the Project\Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select both the Microsoft DataGrid Control entry and the Microsoft DataList Controls entry in the Controls list box.
4. Click on OK to close the Components dialog box.



5. This adds the DataGrid, DataCombo, and DataList control tools to the toolbox; draw those controls as you want them on your form.

Here are the principal data properties you use with these three controls:

- **DataGrid**—**DataSource** = ADO data control's name. You can also set the **AllowAddNew**, **AllowDelete**, **AllowUpdate** properties to True or False to enable or disable those operations.
- **DataCombo**—**DataSource** = ADO data control's name; **DataField** = Name of the field to display in the combo's text box; **ListField** = Name of field to display in the list; **RowSource** = ADO data control's name; and **BoundColumn** = Name of the source field with which you can provide data to another control.
- **DataList**—**DataSource** = ADO data control's name; **DataField** = Name of the field to display in the current selection, **ListField** = Name of field to display in the list, **RowSource** = ADO data control's name, **BoundColumn** = Name of the source field with which you can provide data to another control.

Let's see an example. In this case, we've added an ADO data control and the three ADO data-bound controls to a program, as shown in Figure 11.1, and connected them to the ADO data control using their various properties. The code for this example is to locate in the dbcontrols2 folder on this book's accompanying CD-ROM.

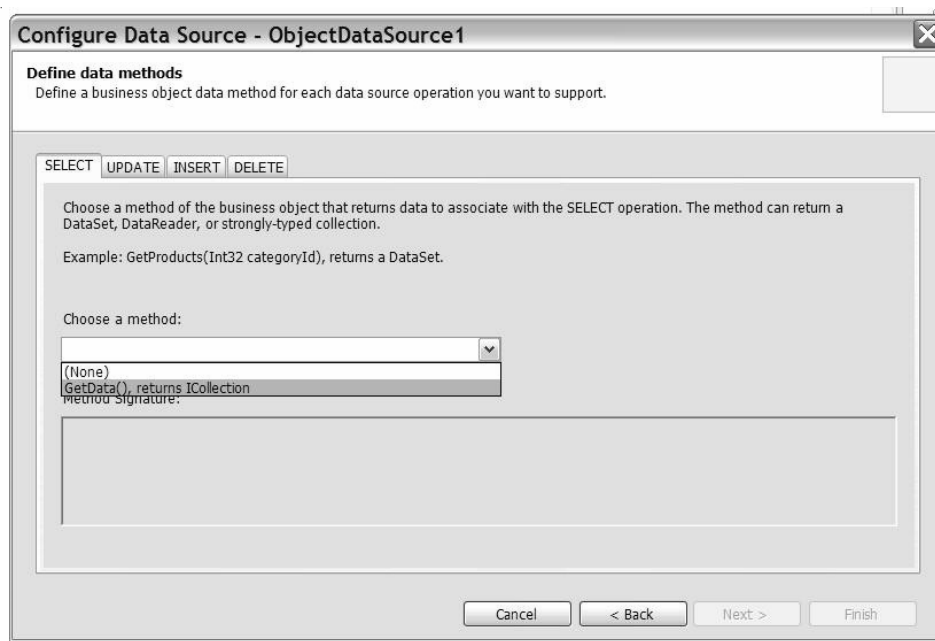


Figure 11.1: Using the ADO Bound Controls.



Did u know?

Error Event Procedure Code Not a Replacement for Your Own Error Handlers  
the ADO Data Control's Error event does not free you from writing error handlers in your own data manipulation code! The Error event handles errors that are not caused by your code. You still must handle the runtime errors generated by your code.

## **11.4 Data Base Access Using the ADO Data Control**

Like a Data Environment, the ADO Data Control also simplifies, automates, or even eliminates some data programming tasks. It has the following similarities to the Data Environment Designer:

Both the Data Environment and the ADO Control expose a Recordset to the programmer.

Both the Data Environment and the ADO Control are used to bind VB controls (such as DataGrid or TextBox controls) to a Recordset.

Both the Data Environment and the ADO Control enable you to determine the Recordset's cursor type, cursor location, locking strategy, and cache size.

When necessary, the programmer can bypass the automated user interface and directly manipulate the Recordset in code. Recordset manipulation is the same in code for both the ADO Data Control and the Data Environment, with but a single syntactic difference: You refer to the ADO Data Control's Recordset with the following syntax:

`ADDataControlName.Recordset`

You refer to the Data Environment's Recordset with this syntax:

`DataEnvironmentName.rsCommandName`

The ADO Data Control differs in the following ways from a Data Environment, however:

The ADO Data Control supports only one Recordset at a time.

The ADO Data Control does not directly expose Command or Connection objects.

The ADO Data Control is visible at runtime and furnishes a visual navigation interface to the user. The following sections discuss how to set up an ADO Data Control, how to manipulate it programmatically, and how to bind controls to its Recordset.

### **11.4.1 Setting Up the ADO Data Control**

To create an ADO Data Control that exposes a Recordset in your application, at the minimum you need to do the following:

Specify a Connection by filling in the `ConnectionString` property.

Specify how to derive a Recordset by setting the `RecordSource` property (which is a complex property requiring its own dialog box to set up).

The detailed steps are as follows:

### **11.4.2 Creating an ADO Data Control**

1. Add the Microsoft ADO DataControl 6.0 (OLEDB) from the Project, Components menu dialog box, as in Figure 11.2. The ADO Data Control icon should now appear in the VB toolbox.
2. Place an instance of the ADO Data Control on the form (see Figure 11.3).



Figure 11.2: Adding the Microsoft ADO Data Control to Your Project's Components.

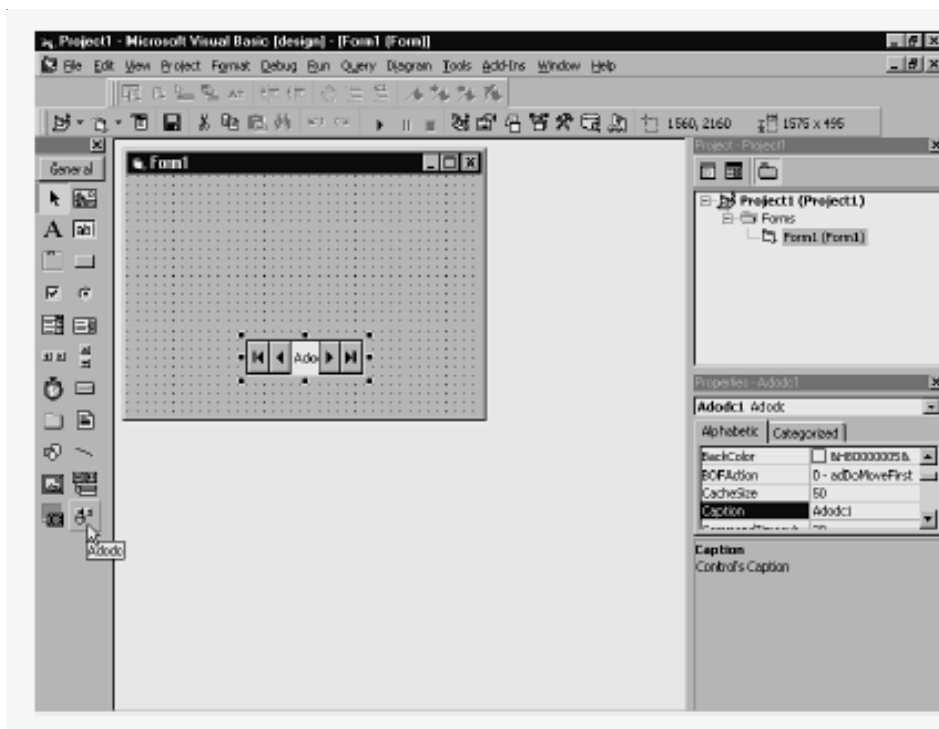
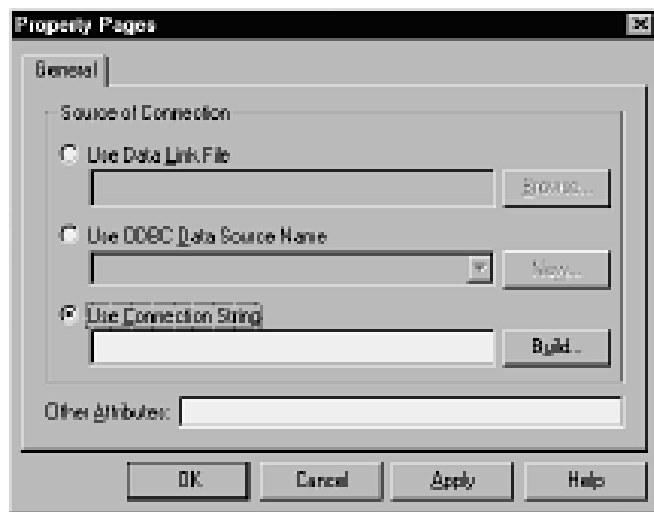


Figure 11.3: Placing an instance of the ADO Data Control on a Form.

3. Change the control's Name and Caption from their default values. (The Caption is for information only, so you can set it to whatever you think will be most informative for the user.)
4. Set the ConnectionString property using steps 5–9.
5. Click the ellipsis next to the ConnectionString property in the ADO Data Control's Properties window to bring up the Property Page dialog box for this property, as shown in Figure 11.4.



**Figure 11.4:** The first and only Property Page dialog box for the ADO Data Control's ConnectionString Property.

6. As Source of Connection, choose one of the following three options:
  - **Use Data Link File.** If you choose this option, you will be able to click the Browse button to specify an existing \*.UDL file).
  - **Use ODBC Data Source Name.** If you choose this option, you will be able to choose an existing ODBC DSN from the drop-down list, or you can create a new DSN by clicking the New button.
  - **Use Connection String.** If you choose this option, you will be able to click the Build button to bring up the Data Link Properties tabbed dialog box.

The following steps assume that you have chosen this option.

7. On the Provider tab of the Data Link Properties tabbed dialog box, choose an OLE DB data provider, such as Microsoft Jet 3.51 OLE DB (see Figure 11.5).

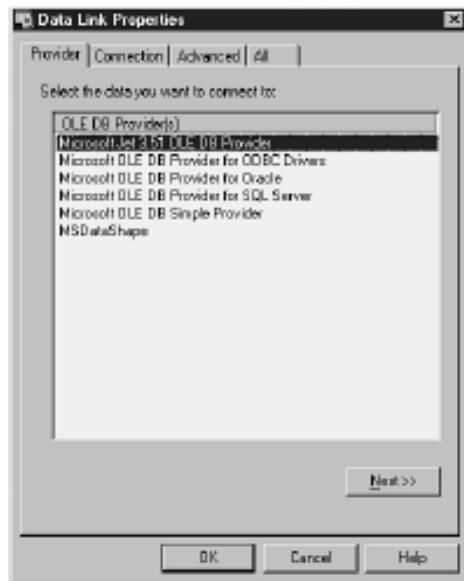


Figure 11.5: Choosing a Provider for the ADO Data Control's Connection String Property.

8. The Connection tab of the Data Link Properties tabbed dialog box will vary in appearance, depending on the provider specified in the preceding step. In the case of the Microsoft Jet 3.51 OLE DB, you are prompted to choose an Access data file and set some security options (see Figure 11.6).

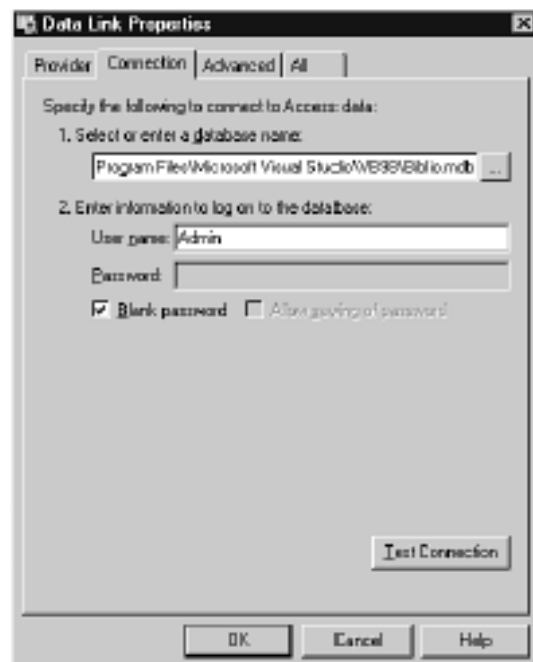


Figure 11.6: Setting up Connection Information for the Jet Provider.

9. Click OK to accept theConnectionString options you have built.
10. Still in the ADO Data Control's Properties window, navigate to the RecordSource property and click the ellipsis button.
11. On the RecordSource tab (see Figure 11.7) of the resulting Property Page dialog box, choose the CommandType (adCmdUnknown, adCmdText, adCmdTable, adCmdStoredProc).

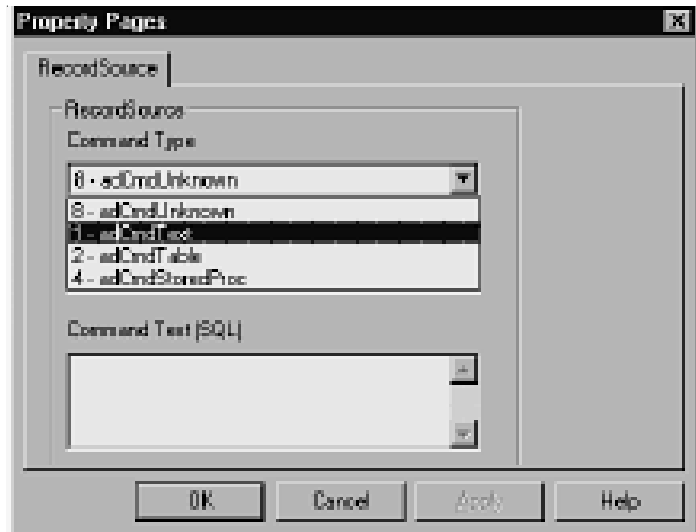


Figure 11.7: The Property Page Dialog Box for an ADO Data Control's RecordSource Property.

12. Complete the dialog box appropriately for the CommandType that you chose:
  - If you chose adCmdText, fill in the text of a valid Select statement in the Command Text field (see Figure 11.8).

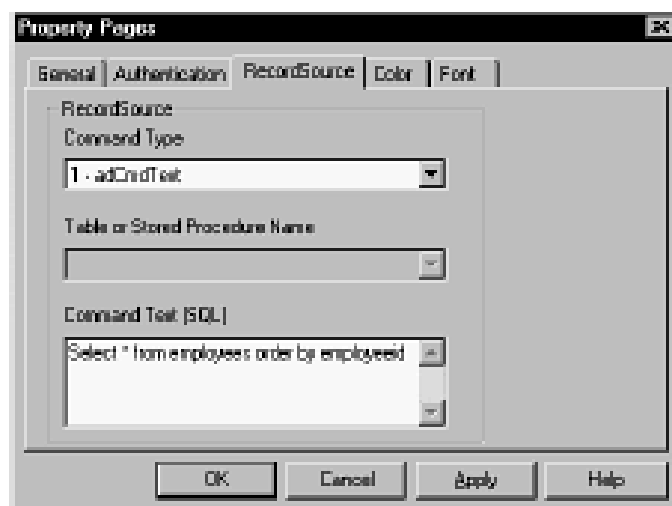


Figure 11.8: A Valid Select Statement as Command Text for the RecordSource Property.

- If you chose adCmdTable or adCmdStoredProc, fill in the appropriate table or stored procedure name in the Table or Stored Procedure Name drop-down list (see Figure 11.9).

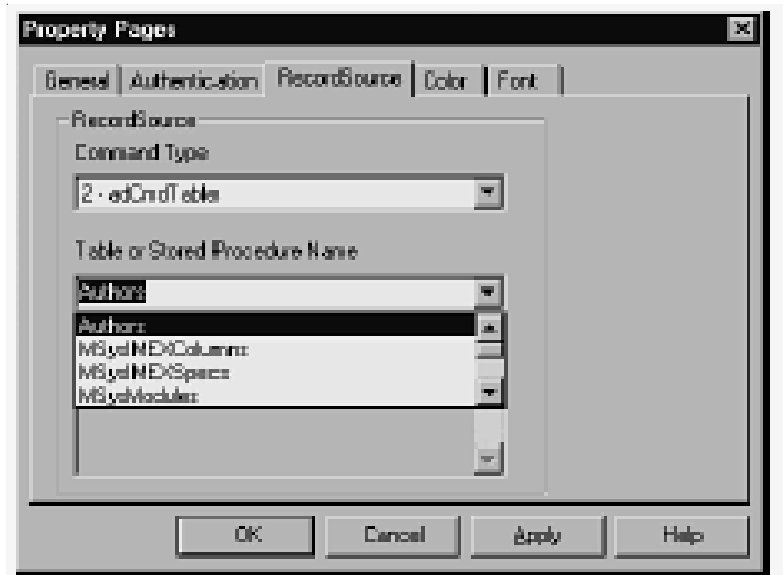


Figure 11.9: A Table or Stored Procedure Name for the RecordSource Property.

13. Click OK to end the RecordSource dialog box.

## 11.5 ADO and ADOCE

ADO is Microsoft's strategic, high-level interface to all kinds of data. An application that uses ADO can access and manipulate data in a database server through an OLE database provider. The primary benefits of ADO are ease of use, high speed, low memory overhead, and a small disk footprint. ADO provides consistent, high-performance access to data for creating a front-end database client or middle-tier business object using an application, tool, language, or even an Internet browser.

ADO is the central component of the Microsoft Universal Data Access strategy. Universal Data Access provides high-performance access to a variety of information sources, including relational and non relational, and an easy-to-use programming interface that is tool- and language-independent. These technologies enable corporations to integrate diverse data sources, create easy-to-maintain solutions, and use their choice of the best tools, applications, and platform services.

ADOCE provides a subset of ADO for the Windows CE operating system that includes implementation of the **Recordset** and **Field** objects. ADOCE adds new database functionality to Windows CE by enabling access to databases stored locally on a device and provides synchronization of data to a network database. ADOCE provides access to the Windows CE database engine from any COM-capable environment, such as the Microsoft Windows CE Toolkit for Microsoft Visual Basic® version 6.0.

ADOCE provides the following features:

- Automatic two-way synchronization to the desktop host.
- New data types: double and Boolean (compared to Windows CE database).
- SQL support. The advanced SQL support included in ADOCE is a subset of the American National Standards Institute (ANSI) SQL. It adds new database engine functionality that provides field names, multiple sort keys, complex filtering, table and index creation, deletion and editing, and join capabilities between multiple tables.
- Familiar recordset.fieldname access.
- Desktop ADO compatibility.



*Case Study*

**A History of Microsoft Access**

Microsoft Access is a full service database program that has been in development since 1992. While it is not as robust as other corporate database platforms, its usability makes it ideal for organizing small data sets, making it a great choice for personal and small business use. Another advantage is that it is quite inexpensive, making it competitive comparable to the heavy duty competition. Compared to most of the other programs in the Microsoft office suite, Access is one of the lesser used ones, although it does have its applications in certain business environments.

The first version of Access was released in late 1992. The software worked well with relatively small data sets, but when the amount of information started to get much larger, data corruption and slowness became a serious issue. Part of the problem was the rapidly increasing amounts of computer memory becoming available at the time—the strength of the hardware available outpaced the skill of software developers, and customer expectations were as such difficult to predict.

As the program continued to improve, more features were added. Access became a development platform for relatively small scale programs, making it a useful tool even for programmers, amateur of otherwise, because of its usage as a launching platform for small but handy applications.

The changing needs of businesses rapidly made older versions of Access obsolete. The data stored in older versions of the program cannot be read by versions of software released after Access 97. Most databases were converted over the years to newer versions to prevent year 2000 related bugs from surfacing.

The various graphical wizards are very useful for newer users to develop database queries. So long as there isn't much data involved and there aren't many simultaneous users, Access is a good business solution. If the data becomes more important later on, it is possible to use the Access "upsizing" feature to upgrade it to the Microsoft SQL Server system.



**Version History**

- 1992 – Access 1.1
- 1993 – Access 2.0
- 1995 – Access 95
- 1997 – Access 97
- 1999 – Access 2000
- 2001 – Access 2002
- 2003 – Access 2003
- 2007 – Access 2007

**Questions**

1. Explain the concept of data base access.
2. Give the all some description of MS access 2007.

**11.6 Summary**

- ADO provides a new and powerful means of accessing data stored in database.
- ADO's object model is simpler than data access objects.
- The remote data control also simplifies, automates, or even eliminates some data programming tasks.
- Data environment and ADO control expose a recordset to the programmer.
- ADOCE provides a subset to ADO for the windows CE operating system that include implementation of the recordset and field objects.

**11.7 Keywords**

**Connection string:** A *connection string* is a string version of the initialization properties needed to connect to a data store and enables you to easily store connection information within your application or to pass it between applications. Without a connection string, you would be required to store or pass a complex array of structures to access data. When accessing OLE DB directly, using `IDataInitialize`, the connection string is passed as a parameter to create an OLE DB data source object.

**Data source:** A *data source* is a readily accessible object that provides data to any data consumer (any class or control that can be bound to a source of external data).

**RecordSource :**The `RecordSource` property is useful if you want to create a reusable form or report. For example, you could create a form that incorporates a standard design, then copy the form and change the `RecordSource` property to display data from a different table, query, or SQL statement.

**SQL:** SQL is short for Structured Query Language and is a widely used database language, providing means of data manipulation (store, retrieve, update, delete) and database creation.

**Value property:** The `Value` property makes it easy to access the value of the first element in a collection of `XElement` objects. This property first checks whether the collection contains at least one object. If the collection is empty, this property returns `Nothing`.



Lab Exercise

1. How to Create an ADO Data Control?

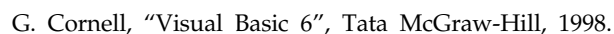
## 11.8 Self Assessment Questions

1. You can set the contents of an ADO Recordset field named ID with the following code:
  - (a) `sClients.ID.Value = strID`
  - (b) `rsClients.Fields(ID).Value = strID`
  - (c) `rsClients!Fields!(ID).Value = strID`
  - (d) `rsClients!ID = strID`
2. The ADO Errors collection
  - (a) Clears before every ADO action.
  - (b) Contains a history of all errors during this session.
  - (c) Contains the last errors generated by an ADO action.
  - (d) Contains a lookup list of all possible ADO error codes.
3. Which code would appropriately display ADO errors (connNWind is an ADO Connection and rsEmployees is an ADO Recordset object)?
  - (a) `Dim adoErr As ADODB.Error For Each adoErr In connNWind.Errors MsgBox adoErr.Description Next /`
  - (b) `Dim adoErr As ADODB.Error For Each adoErr In rsEmployees.Errors MsgBox adoErr.Description Next adoErr`
  - (c) `Dim adoErr As ADODB.Error For Each adoErr In connNWind MsgBox connNWind.Error.Description Next adoErr`
  - (d) `Dim adoErr As Error For Each adoErr In connNWind.Errors MsgBox adoErr.Description Next adoErr`
4. You write the following code to search for a record in an ADO Recordset: `Dim sFindCriterion As String s FindCriterion = "LastName like '" & _txtLastNameToFind & "'" rsEmployees.MoveFirst rsEmployees.Find sFindCriterion` To test whether a record was encountered, you should write the following code:
  - (a) `If rsEmployees.NoMatch Then`
  - (b) `If rsEmployee.EOF Then`
  - (c) `If rsEmployee.BookMark = 0 Then`
  - (d) `If rsEmployee.BookMark = rsRmployee.RecordCount Then`

- ## 11.9 Review Questions

- ## Answers for Self Assessment Questions

- ### 11.10 Further Readings



Online link

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 12: ADO Data Control**

### **CONTENTS**

Objectives

Introduction

12.1 Recordset and Field Objects

12.1.1 Using ADOCE

12.1 ConnectionString Properties

12.3 Displaying Data from Database in Grid

12.3.1 Conceptual Difference

12.3.2 Code Changes for the DataGrid Control

12.4 DataGrid Control Property, Method, and Event Equivalencies

12.4.1 Properties

12.4.2 Methods

12.4.3 Events

12.5 Summary

12.6 Keywords

12.7 Self Assessment Questions

12.8 Review Questions

12.9 Further Readings

### **Objectives**

*After studying this unit, you will be able to:*

- Explain the recordset and field objects.
- Discuss connection string properties.
- Displaying data from database in grid.
- Discuss data grid control property, method.
- Explain event equivalencies.

### **Introduction**

The ADO data control presents a fast and easy way to create a bound form by providing built-in functions that define, navigate, display, add to, and update the records in a recordset.

The resulting form makes it easy for a user to maintain records. Although the data control defines the record set, you'll still need a control for each field to automatically display the bound records.

## 12.1 Recordset and Field Objects

The ADOCE control has two objects: **Recordset** and **Field**. A recordset is a virtual database table whose fields and rows correspond to a subset of the fields and rows in an actual database table on the Windows CE-based device. When you make additions, deletions, or changes to the information in a recordset row, you can pass those changes to the corresponding parts of the table. When you change data in the recordset, the recordset stores the changes in memory, enabling you to cancel them before the underlying database is updated. ADOCE does not support batch updates. Only one row at a time can have data that is changed but not committed to the underlying database.

Table 12.1: Methods that the Recordset object supports

| Method              | Description                                                  |
|---------------------|--------------------------------------------------------------|
| <b>AddNew</b>       | Inserts a new row into the recordset.                        |
| <b>CancelUpdate</b> | Cancels changes held in memory.                              |
| <b>Clone</b>        | Duplicates a recordset.                                      |
| <b>Close</b>        | Closes a recordset.                                          |
| <b>Delete</b>       | Deletes a row from the recordset.                            |
| <b>GetRows</b>      | Returns data stored in the recordset.                        |
| <b>Move</b>         | Changes the pointer to the active row in the recordset.      |
| <b>MoveFirst</b>    | Makes the first row active.                                  |
| <b>MoveLast</b>     | Makes the last row active.                                   |
| <b>MoveNext</b>     | Moves the active row pointer to the next row.                |
| <b>MovePrevious</b> | Moves the active row pointer to the previous row.            |
| <b>Open</b>         | Defines and opens recordsets; runs SQL commands.             |
| <b>Supports</b>     | Determines if the recordset supports certain features.       |
| <b>Update</b>       | Commits changes held in memory and updates the actual table. |

Table 12.2: Properties that the Recordset object supports

| Property                | Description                                                                                                                              |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AbsolutePage</b>     | Specifies which page to move for a new current record.                                                                                   |
| <b>AbsolutePosition</b> | Specifies the ordinal position of a <b>Recordset</b> object's current record.                                                            |
| <b>ActiveConnection</b> | Sets the current database connection. Always a zero-length string (""), For H/PC running Pro Edition software, the name of the cdb file. |
| <b>BOF</b>              | Indicates whether the current record position is before the first record in a <b>Recordset</b> object.                                   |
| <b>Bookmark</b>         | Specifies a bookmark that uniquely identifies a record in a <b>Recordset</b> object.                                                     |
| <b>CacheSize</b>        | Specifies the number of records from a <b>Recordset</b> object that are cached locally in memory.                                        |

|                    |                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------|
| <b>CursorType</b>  | Indicates the type of cursor used in a <b>Recordset</b> object.                                   |
| <b>EditMode</b>    | Indicates the editing status of the current record.                                               |
| <b>EOF</b>         | Indicates that the current record position is after the last record in a <b>Recordset</b> object. |
| <b>LockType</b>    | Indicates the type of locks placed on records during editing.                                     |
| <b>PageCount</b>   | Indicates how many pages of data the <b>Recordset</b> object contains.                            |
| <b>PageSize</b>    | Indicates how many records constitute one page in the recordset.                                  |
| <b>RecordCount</b> | Returns a Long value that indicates the current number of records in a <b>Recordset</b> object.   |
| <b>Source</b>      | Indicates the source for the data in a <b>Recordset</b> object—SQL statement or table name.       |

**Field** objects should not be directly created because they exist only in the context of an existing recordset. Use the **Set** function to refer to a specific **Field** object. The **Field** object has no methods or events. With the exception of the **Value** property, all the properties are read-only.

**Table 12.3: Properties that the Field object supports**

| Property               | Description                                                                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ActualSize</b>      | Indicates the actual length, in bytes, of a field's value.                                                                                                                       |
| <b>Attributes</b>      | Returns a value that indicates one or more characteristics of a <b>Field</b> object.                                                                                             |
| <b>DefinedSize</b>     | Used to determine the data capacity of a <b>Field</b> object. It returns the defined size, in characters, of the field Compare with ActualSize, which returns the size in bytes. |
| <b>Name</b>            | Returns the name of a field.                                                                                                                                                     |
| <b>Type</b>            | Indicates the data type of a <b>Field</b> object.                                                                                                                                |
| <b>UnderlyingValue</b> | Indicates a <b>Field</b> object's current value in the database.                                                                                                                 |
| <b>Value (default)</b> | Indicates a <b>Field</b> object's current value in the recordset.                                                                                                                |

The **Fields** collection contains a **Field** object for each column in the recordset. You can refer to a particular field by name or by index. The **Fields** collection supports the **Count** property, which indicates the number of fields in a recordset.

### 12.1.1 Using ADOCE

With ADOCE, you can move databases to and from your Windows CE-based device. You can also create and access databases on the device and on an emulator on a desktop computer. The following sections describe how to use ADOCE and show examples. Note that ControlConsts.bas is installed in \Program Files\Microsoft Visual Studio\VB98\VBCE\Samples. This file contains the definitions for the enumerations in ADOCE and other controls



*Did u know?*

1. Other Methods for Opening a Recordset The section titled “Accessing Data with the Execute Direct Model” in the following chapter discusses how to use the Execute method of ADO Connection and Command objects to open a Recordset.

2. Additional ADO Data Control Properties That Affect the Recordset You will eventually need to fine-tune the Recordset's behavior by setting the `CursorLocation`, `CursorType`, and `Locktype` properties. These properties correspond to the standalone Recordset object's properties of the same name. The next chapter discusses these properties and their meaning in more detail.

## 12.2 ConnectionString Properties

Use the **ConnectionString** property to specify a data source by passing a detailed connection string containing a series of *argument = value* statements separated by semicolons.

ADO supports five arguments for the **ConnectionString** property; any other arguments pass directly to the provider without any processing by ADO. The arguments ADO supports are as follows:

| Argument         | Description                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Providers=       | Specifies the name of a provider to use for the connection.                                                                           |
| File Name=       | Specifies the name of a provider-specific file (for example a persisted data source object) containing preset connection information. |
| Remote Provider= | Specifies the name of a provider to use when opening a client-side connection (Remote Data Service only)                              |
| Remote Service=  | Specifies the path name of the server to use when opening a client-side connection (Remote Data Service only)                         |
| URL=             | Specifies the connection string as an absolute URL identifying a resource such as a file or directory.                                |

After you set the **ConnectionString** property and open the Connection object, the provider may alter the contents of the property, for example, by mapping the ADO-defined argument names to their equivalents for the specific provider.

**Table 12.4: Lists the Default ADO Provider for Each Windows Operating System**

| Default ADO provider                                                                                                                                                                                                                                    | Windows operating system                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MSDASQL<br>(To improve the readability of source code, explicitly specify the provider name in the connection string.)                                                                                                                                  | Windows 2000 (32-bit)<br>Windows XP (32-bit)<br>Windows 2003 Server (32-bit)<br>Windows Vista (32-bit)<br>Windows Vista Service Pack 1 or later (32-bit and 64-bit)<br>Windows versions after Windows Vista (32-bit and 64-bit) |
| No default.<br>When an ADO application runs on the following operating systems and does not specify the provider explicitly, ADO returns the following error: "ADODB.Connection: provider is not specified and there is no designated default provider" | Windows 2000 (64-bit)<br>Windows XP (64-bit)<br>Windows 2003 Server (64-bit)<br>Windows Vista (64-bit)                                                                                                                          |

The **ConnectionString** property automatically inherits the value used for the *ConnectionString* argument of the Open method, so you can override the current **ConnectionString** property during the **Open** method call.

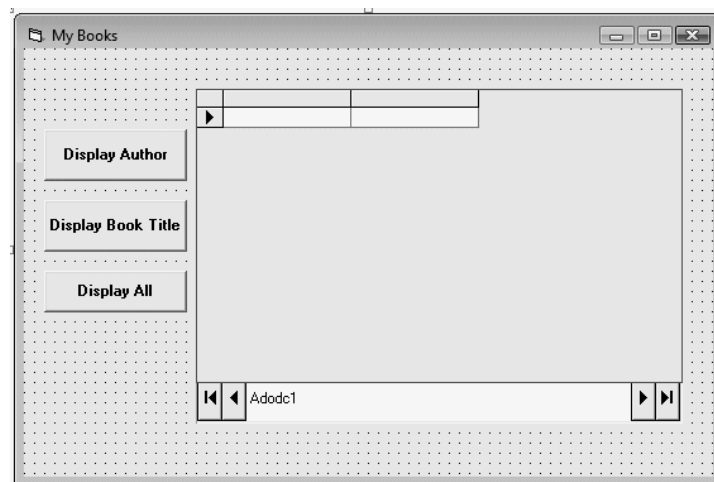
Because the *File Name* argument causes ADO to load the associated provider, you cannot pass both the *Provider* and *File Name* arguments.

The **ConnectionString** property is read/write when the connection is closed and read-only when it is open.

Duplicates of an argument in the **ConnectionString** property are ignored. The last instance of any argument is used.

### **12.3 Displaying Data from Database in Grid**

We will start Visual Basic and insert an ADO control, a DataGrid and three command buttons. Name the three command buttons as **cmdAuthor**, **cmdTitle** and **cmdAll**. Change their captions to **Display Author**, **Display Book Title** and **Display All** respectively. The design interface is shown in Figure 12.1.



**Figure 12.1:** Displaying Data from Database in Grid.

Now you need to connect the database to the ADO data control. However, you need to make one change. At the ADODC property pages dialog box, click on the Recordsource tab and select **1-adCmdText** under command type and under Command Text(SQL) key in **SELECT \* FROM book**.

Next, click on the command button **cmdAuthor** and key in the following statements:

```
Private Sub cmdAuthor_Click()
Adodc1.RecordSource = "SELECT Author FROM book"
Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource
End Sub
```

and for the command button **cmdTitle**, key in

```
Private Sub cmdTitle_Click()
Adodc1.RecordSource = "SELECT Title FROM book"
```



```

Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource
End Sub

```

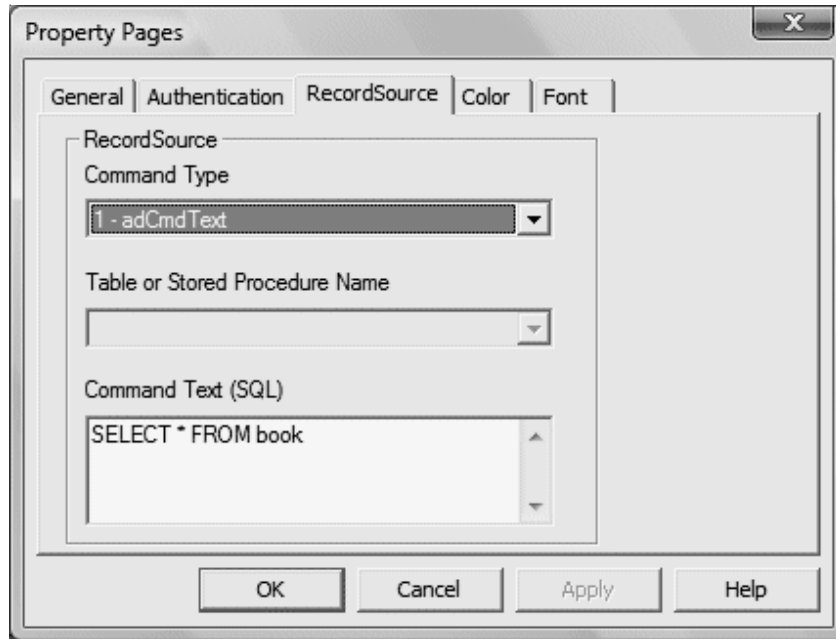


Figure 12.2: Property Page.

Finally for the command button cmdAll, key in

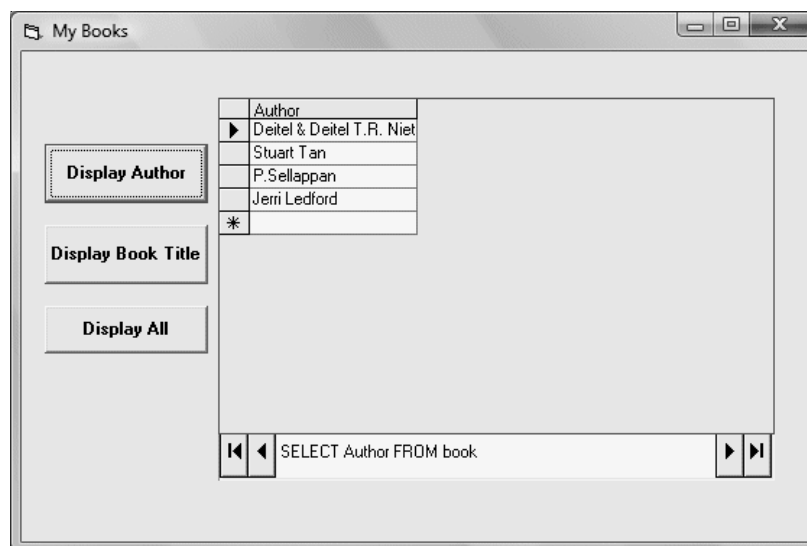


Figure 12.3: Section of Book.

```
Private Sub cmdAll_Click()
Adodc1.RecordSource = "SELECT * FROM book"
Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource
End Sub
```

Now, run the program and when you click on the Display Author button, only the names of authors will be displayed, as shown below:

and when you click on the Display Book Title button, on the book titles will be displayed, as show below:

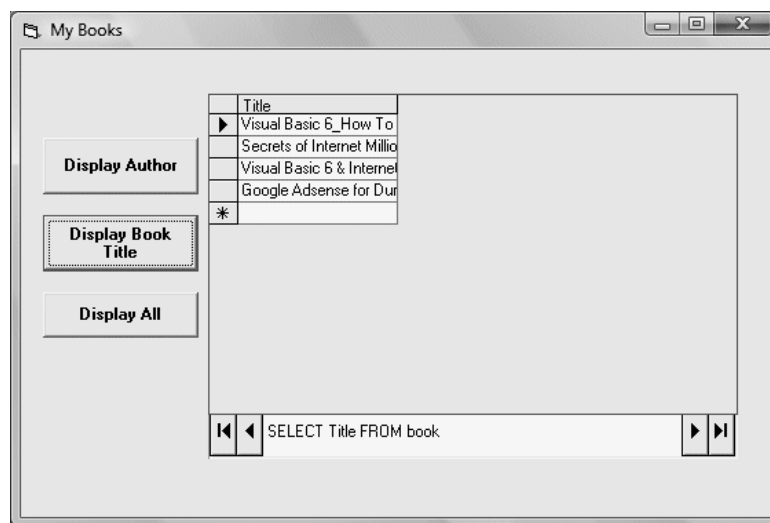


Figure 12.4: Selection of Title.

Lastly, click on the Display All button and all the information will be displayed.

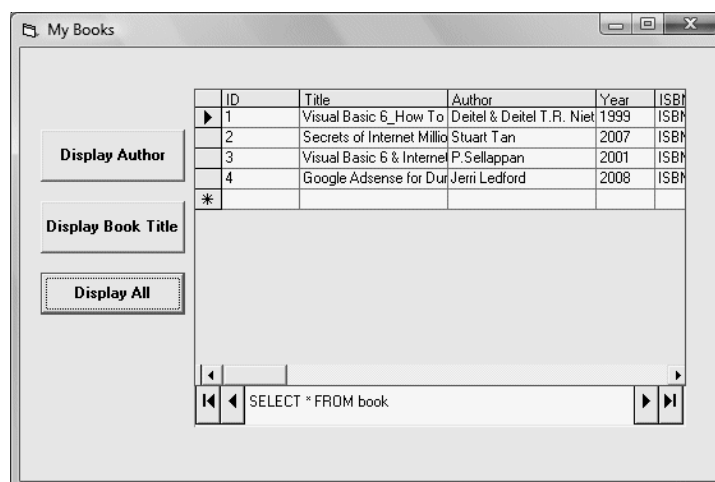


Figure 12.5: View of Data.

## 12.3.1 Conceptual Differences

### 12.3.1.1 Data Binding

The Visual Basic 2005 **DataGridView** control does not need data-specific methods or events because all actions are performed through the data source. Because of this separation of presentation and data functionality, the data source can be changed with or without user interface input. Also, multiple controls bound to the same data source will always stay in sync.

### 12.3.1.2 Navigation

Properties for viewing and navigation in the **DataGrid** control (such as **TabAction**, **EnterAction**, **AllowArrows**, **WrapCellPointer**, and **Scrollable**) are no longer needed. For example, the grid functions as though the **Scrollable** property is set to **True**: If more data exists than can be displayed, a scroll bar appears automatically. Excel-style navigation through the grid is the default, allowing the user to move forward with the TAB key and backward with the SHIFT+TAB key combination.

### 12.3.1.3 Caption Property

In Visual Basic 6.0, the **Caption** property is used to display a title bar above the grid; if the **Caption** property is left empty, no title bar is displayed.

The Visual Basic 2005 **DataGridView** control does not support a title bar. However, you can achieve the same effect using a Label control.

### 12.3.1.4 Data Formatting

In Visual Basic 6.0, formatting data in the **DataGrid** control is handled using the **DataFormat** property and a **StdDataFormat** object. Formatting is applied on a column-by-column basis.

In the Visual Basic 2005 **DataGridView** control, formatting is accomplished using the **Format** property of a **DataGridViewCellStyle** object. Formatting can be applied to individual cells, columns, or rows.

### 12.3.1.5 hWndEditor Property

In Visual Basic 6.0, the **hWndEditor** property is used to pass the window handle assigned to a **DataGrid** control's editing window to a Windows API call.

The Visual Basic 2005 **DataGridView** control does not have a separate window handle when in edit mode; instead, use the **Handle** property of the **DataGridView** control or any embedded edit controls.

### 12.3.1.6 MarqueeStyle Property

In Visual Basic 6.0, the **MarqueeStyle** property controls the appearance of a selected cell or row by changing the border style, inverting the foreground and background colors, or invoking an edit window.

There is no direct equivalent for the Visual Basic 2005 **DataGridView** control. However, you can achieve the same effect through a combination of the **SelectionMode**, **CellBorderStyle**, and **Format** properties.

### 12.3.1.7 SelLength, SelStart, SelText Properties

In the Visual Basic 6.0 **DataGrid** control, when a cell enters edit mode, the **SelLength**, **SelStart**, and **SelText** properties can be used to set the initial position of the caret or to highlight a portion of the text in the cell.

In the Visual Basic 2005 **DataGridView** control, these properties no longer exist. The cells in the **DataGridView** control are based on the **TextBox** control; by adding code to the **EditingControlShowing** event handler you can access the **SelectionLength**, **SelectionStart**, and **SelectedText** properties of the underlying control.

### 12.3.1.8 Split Views

The Visual Basic 6.0 **DataGrid** control supports a split view, allowing the user to display the same data side-by-side. The **Split** object and the **Split**, **Splits**, and **TabAcrossSplits** properties control the ability to display a split view.

In the Visual Basic 2005 **DataGridView** control, these properties no longer exist. However, you can achieve the same effect using one or more **SplitContainer** controls and multiple **DataGridView** controls. To duplicate the functionality of the **TabAcrossSplits** property, you can use the **StandardTab** property of the **DataGridView** control.

## 12.3.2 Code Changes for the DataGrid Control

The following code demonstrates the differences between Visual Basic 6.0 and Visual Basic 2005 by showing how, in each version, to highlight the text in a cell when a user selects the cell in a **DataGridView** control.

```
' Visual Basic 6.0
Private Sub DataGrid1_Click()
 DataGrid1.SelStart = 1
 DataGrid1.SelLength = DataGrid1.Text
 MsgBox("The selected text is " & DataGrid1.SelText)End Sub
' Visual Basic 2005
Private Sub DataGridView1_EditingControlShowing(_
 ByVal sender As Object, ByVal e As System.Windows.Forms. _
 DataGridViewEditingControlShowingEventArgs) _
 Handles DataGridView1.EditingControlShowing
 CType(e.Control, TextBox).SelectionStart = 0
 CType(e.Control, TextBox).SelectionLength = Len(CType(e.Control, _
 TextBox).Text)
 MsgBox("The selected text is " & CType(e.Control, _
 TextBox).SelectedText)
End Sub]
```

## 12.4 DataGrid Control Property, Method, and Event Equivalencies

The following tables list Visual Basic 6.0 properties, methods, and events, along with their Visual Basic 2005 equivalents. Those properties, methods, and events that have the same name and behavior are not listed. All Visual Basic 2005 enumerations map to the System.Windows. Forms namespace unless otherwise noted.

This table provides links to topics explaining behavior differences. Where there is no direct equivalent in Visual Basic 2005, links are provided to topics that present alternatives.

### 12.4.1 Properties

| Visual Basic 6.0           | Visual Basic 2005 Equivalent                                            |
|----------------------------|-------------------------------------------------------------------------|
| <b>AddNewMode</b>          | New implementation. Data operations are handled in the data source.     |
| <b>Align</b>               | Dock and Anchor                                                         |
| <b>AllowAddNew</b>         | AllowNew ( <b>BindingSource</b> )                                       |
| <b>AllowArrows</b>         | No longer an adjustable property; arrow navigation always allowed.      |
| <b>AllowDelete</b>         | AllowRemove ( <b>BindingSource</b> )                                    |
| <b>AllowRowSizing</b>      | AllowUserToResizeRows                                                   |
| <b>AllowUpdate</b>         | AllowEdit ( <b>BindingSource</b> )                                      |
| <b>Appearance</b>          | New implementation.                                                     |
| <b>ApproxCount</b>         | New implementation. Data operations are handled in the data source.     |
| <b>BackColor</b>           | BackgroundColor                                                         |
| <b>Bookmark</b>            | New implementation. You can now directly access any item.               |
| <b>Caption</b>             | New implementation. Use a <b>Label</b> control to simulate a title bar. |
| <b>Col</b>                 | SelectedColumns                                                         |
| <b>ColumnHeaders</b>       | ColumnHeadersVisible                                                    |
| <b>Container</b>           | Parent; inherited from <b>Control</b> .                                 |
| <b>CurrentCellModified</b> | IsCurrentCellDirty, IsCurrentCellInEditMode                             |
| <b>CurrentCellVisible</b>  | CurrentCell                                                             |
| <b>DataChanged</b>         | IsCurrentCellDirty, IsCurrentRowDirty                                   |
| <b>DataFormats</b>         | <b>DataGridViewCellStyle</b> .object.                                   |
| <b>DefColWidth</b>         | Width <b>DataGridViewColumn</b> object.                                 |
| <b>DragIcon</b>            | New implementation.                                                     |
| <b>DragMode</b>            | New implementation.                                                     |
| <b>EditActive</b>          | <b>IsCurrentCellInEditMode</b>                                          |
| <b>FirstRow</b>            | FirstDisplayedScrollingRowIndex                                         |

|                          |                                                                                                                                                    |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Font</b>              |                                                                                                                                                    |
| <b>FontBold</b>          |                                                                                                                                                    |
| <b>FontItalic</b>        |                                                                                                                                                    |
| <b>FontName</b>          | Font                                                                                                                                               |
| <b>FontSize</b>          |                                                                                                                                                    |
| <b>FontStrikethrough</b> |                                                                                                                                                    |
| <b>FontUnderline</b>     |                                                                                                                                                    |
| <b>ForeColor</b>         | ForeColor                                                                                                                                          |
| <b>HeadFont</b>          | <b>DataGridViewCellStyle</b> object.                                                                                                               |
| <b>HeadLines</b>         | No direct equivalent. Use the <b>WrapMode</b> property of the <b>DataGridViewCellStyle</b> object in combination with <b>ColumnHeadersHeight</b> . |
| <b>Height</b>            | Height, inherited from <b>Control</b> class.                                                                                                       |
| <b>HelpContextID</b>     | New implementation.                                                                                                                                |
| <b>hWnd</b>              | <b>Handle</b>                                                                                                                                      |
| <b>hWndEditor</b>        | New implementation. Use <b>Handle</b> .                                                                                                            |
| <b>Index</b>             | New implementation.                                                                                                                                |
| <b>Left</b>              | Left, inherited from <b>Control</b> class.                                                                                                         |
| <b>LeftCol</b>           | <b>FirstDisplayedScrollingColumnIndex</b>                                                                                                          |
| <b>MarqueeStyle</b>      | No direct equivalent. Use the <b>SelectionMode</b> , <b>CellStyle</b> , and <b>Format</b> properties.                                              |
| <b>RecordSelectors</b>   | <b>RowHeadersVisible</b>                                                                                                                           |
| <b>Row</b>               | <b>SelectedRows</b>                                                                                                                                |
| <b>RowDividerStyle</b>   | <b>GridColor</b> , <b>CellStyle</b> , <b>RowHeadersBorderStyle</b> , <b>ColumnHeadersBorderStyle</b> properties.                                   |
| <b>RowHeight</b>         | Height                                                                                                                                             |
| <b>SelBookmarks</b>      | New implementation. You can now directly access any item.                                                                                          |
| <b>SelEndCol</b>         |                                                                                                                                                    |
| <b>SelStartCol</b>       | <b>SelectedCells</b> , <b>SelectedColumns</b>                                                                                                      |
| <b>SelLength</b>         | No direct equivalent. Use the <b>SelectionLength</b> property in the <b>EditingControlShowing</b> event handler.                                   |
| <b>SelStart</b>          | No direct equivalent. Use the <b>SelectionStart</b> property in the <b>EditingControlShowing</b> event handler.                                    |

|                        |                                                                                                               |
|------------------------|---------------------------------------------------------------------------------------------------------------|
| <b>SelText</b>         | No direct equivalent. Use the <b>SelectedText</b> property in the <b>EditingControlShowing</b> event handler. |
| <b>Split Splits</b>    |                                                                                                               |
| <b>TabAcrossSplits</b> | New implementation. Split views are not directly supported; use a <b>SplitContainer</b> control.              |
| <b>TabAction</b>       | <b>StandardTab</b>                                                                                            |
| <b>Tag</b>             | New implementation.                                                                                           |
| <b>Text</b>            | <b>CurrentCell.Value</b>                                                                                      |
| <b>ToolTipText</b>     | ToolTip component                                                                                             |
| <b>Top</b>             | Top                                                                                                           |
| <b>VisibleCols</b>     | <b>DisplayedColumnCount</b> method.                                                                           |
| <b>VisibleRows</b>     | <b>DisplayedRowCount</b> method.                                                                              |
| <b>WhatsThisHelpID</b> | New implementation.                                                                                           |
| <b>Width</b>           | Width, inherited from <b>Control</b> class.                                                                   |
| <b>WrapCellPointer</b> | No longer an adjustable property; default behavior is <b>WrapCellPointer = True</b> .                         |

### 12.4.2 Methods

| Visual Basic 6.0     | Visual Basic 2005 Equivalent                                                                                                 |
|----------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>CaptureImage</b>  | New implementation. Capturing the contents of a <b>DataGridView</b> control to a <b>PictureBox</b> control is not supported. |
| <b>ClearFields</b>   | New implementation. Column formatting is automatic when re-binding.                                                          |
| <b>ClearSelCols</b>  | <b>ClearSelection</b>                                                                                                        |
| <b>ColContaining</b> | <b>IndexOf (DataGridViewColumnCollection)</b>                                                                                |
| <b>Drag</b>          | New implementation.                                                                                                          |
| <b>GetBookmark</b>   | New implementation. Bookmarks are no longer supported.                                                                       |
| <b>HoldFields</b>    | New implementation. Column formatting is preserved when re-binding.                                                          |
| <b>Move</b>          | <b>SetBounds</b> , inherited from <b>Control</b> class.                                                                      |
| <b>Rebind</b>        | <b>ResetBindings</b> , inherited from <b>Control</b> class.                                                                  |
| <b>RowBookmark</b>   | New implementation. Bookmarks are no longer supported.                                                                       |
| <b>RowContaining</b> | <b>IndexOf (DataGridViewColumnCollection)</b>                                                                                |
| <b>RowTop</b>        | <b>GetContentBounds (DataGridViewCell)</b>                                                                                   |

|                        |                                                                                                  |
|------------------------|--------------------------------------------------------------------------------------------------|
| <b>Scroll</b>          | New implementation method. Use the <b>CurrentCell</b> property.                                  |
| <b>SetFocus</b>        | Focus                                                                                            |
| <b>ShowWhatsThis</b>   | New implementation.                                                                              |
| <b>SplitContaining</b> | New implementation. Split views are not directly supported; use a <b>SplitContainer</b> control. |
| <b>ZOrder</b>          | <b>BringToFront()</b> or <b>SendToBack()</b> functions                                           |

---

### 12.4.3 Events

---

| Visual Basic 6.0       | Visual Basic 2005 Equivalent                                        |
|------------------------|---------------------------------------------------------------------|
| <b>AfterColEdit</b>    | CellEndEdit                                                         |
| <b>AfterColUpdate</b>  |                                                                     |
| <b>AfterDelete</b>     | New implementation. Data operations are handled in the data source. |
| <b>AfterUpdate</b>     | RowsAdded                                                           |
| <b>BeforeColEdit</b>   | CellBeginEdit                                                       |
| <b>BeforeColUpdate</b> |                                                                     |
| <b>BeforeDelete</b>    |                                                                     |
| <b>BeforeInsert</b>    |                                                                     |
| <b>BeforeUpdate</b>    | New implementation. Data operations are handled in the data source. |
| <b>ButtonClick</b>     | Click ( <b>Button</b> control)                                      |
| <b>Change</b>          | TextChanged                                                         |
| <b>Click</b>           | SelectedIndexChanged                                                |
| <b>ColEdit</b>         | <b>CellBeginEdit</b>                                                |
| <b>ColResize</b>       | ColumnWidthChanged                                                  |
| <b>DblClick</b>        | CellMouseDoubleClick                                                |
| <b>DragDrop</b>        |                                                                     |
| <b>DragOver</b>        | New implementation.                                                 |
| <b>Error</b>           | DataError                                                           |
| <b>HeadClick</b>       | ColumnHeaderMouseClick                                              |
| <b>MouseDown</b>       | CellMouseDown                                                       |
| <b>MouseMove</b>       | CellMouseMove                                                       |
| <b>MouseUp</b>         | CellMouseUp                                                         |
| <b>OLECompleteDrag</b> |                                                                     |
| <b>OLEDragDrop</b>     |                                                                     |
| <b>OLEDragOver</b>     |                                                                     |



OLEGiveFeedback

OLESetData

OLEStartDrag      New implementation.

OnAddNew      **RowsAdded**

RowColChange      CurrentCellChanged

RowResize      RowHeightChanged

SelChange      SelectionChanged

SplitChange      New implementation. Split views are not directly supported; use a **SplitContainer** control.

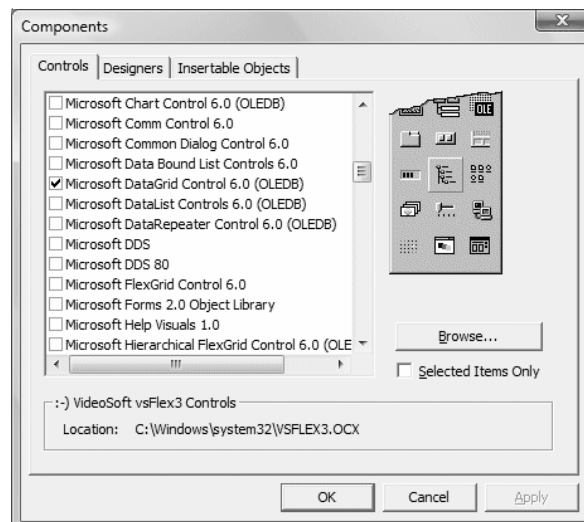
Validate      Validating

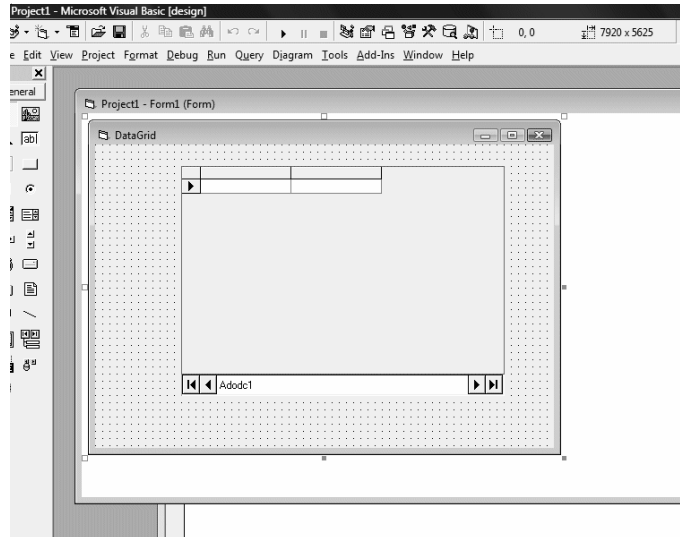


## Case Study

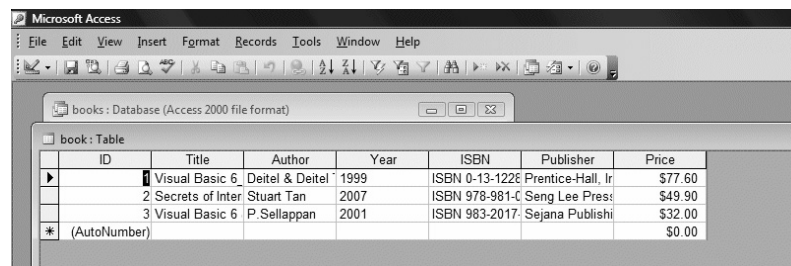
## Using Microsoft DataGrid Control 6.0

DataGrid control is not the default item in the Visual Basic control toolbox; you have to add it from the VB6 components. To add the DataGrid control, click on the project in the menu bar and select components where a dialog box that displays all the available VB6 components. Select Microsoft DataGrid Control 6.0 by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the Microsoft ADO data control so that you are able to access the database. Lastly, click on the OK button to exit the dialog box. Now you should be able to see that the DataGrid control and the ADO data control are added to the toolbox. The next step is to drag the DataGrid control and the ADO data control into the form. The components dialog box is shown below:



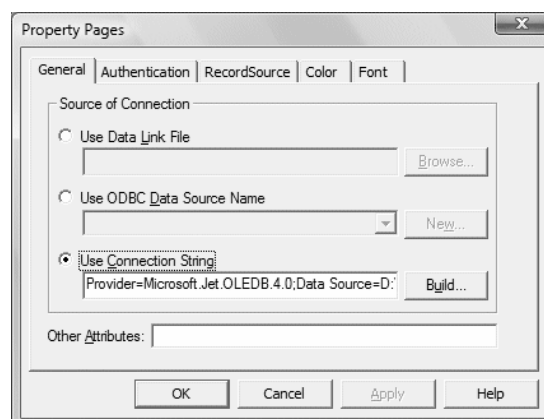


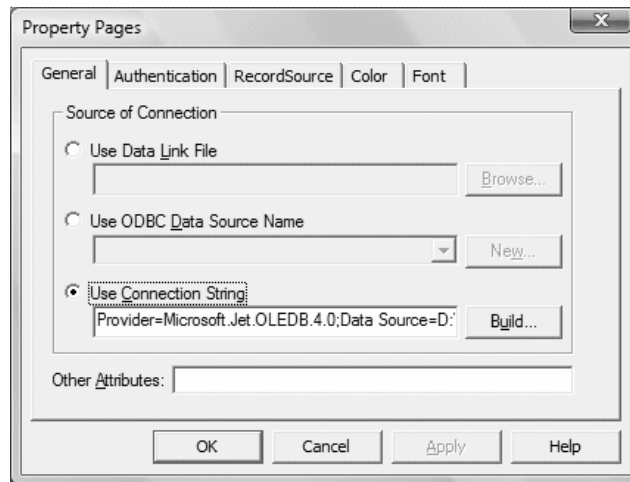
Before we proceed, we need to create a database file using Microsoft Access. Here we have created a file to store the information of books and we name the table book. After we have created the table. The table is shown below:



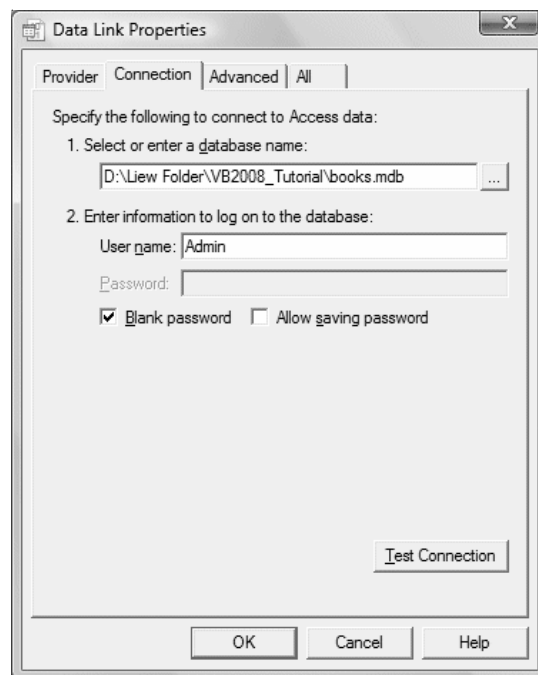
| ID | Title            | Author          | Year | ISBN           | Publisher         | Price   |
|----|------------------|-----------------|------|----------------|-------------------|---------|
| 1  | Visual Basic 6   | Deitel & Deitel | 1999 | ISBN 0-13-1226 | Prentice-Hall, Ir | \$77.60 |
| 2  | Secrets of Inter | Stuart Tan      | 2007 | ISBN 978-981-0 | Seng Lee Pres     | \$49.90 |
| 3  | Visual Basic 6   | P. Sellappan    | 2001 | ISBN 983-2017  | Sejana Publishi   | \$32.00 |
| *  | (AutoNumber)     |                 |      |                |                   | \$0.00  |

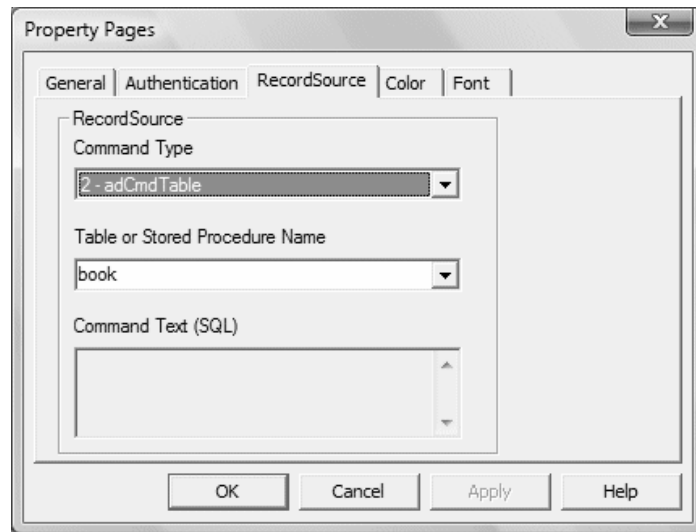
Now you need to connect the database to the ADO data control. To do that, right click on the ADO data control and select the ADODC properties, the following dialog box will appear



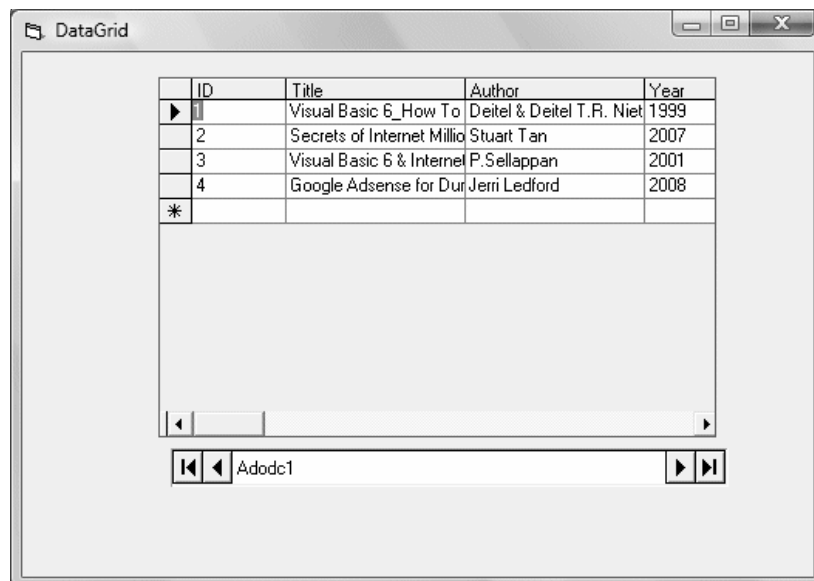


Next click on the Build button and the Data Link Properties dialog box will appear (as shown below). In this dialog box, select the database file you have created. Press test connection to see whether the connection is successful. If the connection is successful, click OK to return to the ADODC property pages dialog box. At the ADODC property pages dialog box, click on the Recordsource tab and select 2-adCmdTable under command type and select book as the table name, then click OK.





Finally, you need to display the data in the DataGrid control. To accomplish this, go to the properties window and set the DataSource property of the DataGrid to Adodc1. You can also permit the user to add and edit your records by setting the AllowUpdate property to True. If you set this property to false, the user cannot edit the records. Now run the program and the output window is shown below:



#### Questions

1. What is DataGrid view?
2. Explain ADODC property.

## 12.5 Summary

- In record and field objects when you changed data in the recordset, the recordset stores the changes in memory.
- With the help of ADOCE you can move database to end from your windows CE-based device.
- Connecting string property pass directly to the provider without any processing by ADO.
- Because of the separation of presentation and data functionality, the data source can be changed with or without user interface input is data printing.
- Caption property is used to display a title bar above the grid.
- In data formatting, formatting can be applied to individual cells, columns and rows.

## 12.6 Keywords

**CommandType:** CommandType property is set to StoredProcedure, you should set the CommandText property to the name of the stored procedure. The command then executes this stored procedure when you call one of the Execute methods (for example, ExecuteReader or ExecuteNonQuery).

**DataGridView:** The Visual Basic *DataGridView* control provides a table in which rows and columns from a database table can be displayed and modified. In this chapter we will explore the steps necessary to build a DataGridView into a Visual Basic application and connect it to a database table.

**Fields:** fields are members of a class, also known as variables. A programmer creates fields at any level of his projects to store and access data within it.

**Recordset.** The ADO Recordset object is used to hold a set of records from a database table. A Recordset object consist of records and columns (fields).



Lab Exercise

1. How to Create an ADO Data Control?

## 12.7 Self Assessment Questions

1. You create a Connection object named connMy and a Command object named cmdMy in a Data Environment Designer named deMy. In your code, you want to programmatically write changes to the underlying data. The line of code that you need to write would be:
  - (a) deMy.rsconnMy.Update
  - (b) deMy.cmdMy.rs.Update
  - (c) deMy.connMy.rscmdMy.Update
  - (d) deMy.rscmdMy.Update

2. How can you place a DataGrid on a form that is automatically bound to the Recordset of a Data Environment's Command object? (Select all that apply.)
  - (a) Drag the Command object with the right mouse button from the Data Environment to the form.
  - (b) Drag the Command object with the left mouse button from the Data Environment to the form.
  - (c) Highlight the Command object with the mouse and then right-click the form.
  - (d) Highlight the Command object with the mouse and then left-click the form.
3. The ADO Data Control's RecordSource property
  - (a) Exposes a Recordset
  - (b) Contains the settings for creating a Recordset
  - (c) Exposes a Fields collection
  - (d) Exposes a Command object
4. A VB control's DataMember property refers to
  - (a) A Recordset object
  - (b) A Connection object.
  - (c) A Command object
  - (d) A Data Environment
5. Setting the adStatus parameter of an ADO Connection object's WillConnect event procedure to a value of adStatusUnwantedEvent
  - (a) Causes the compiler to ignore this event procedure.
  - (b) Causes the event procedure to immediately terminate.
  - (c) Causes a runtime error if there is no error handler written in the event procedure.
  - (d) Causes the event procedure to run only once during the application session.

## **12.8 Review Questions**

1. Which property of a Recordset object should you check to see if the preceding call to the find method was successful?
2. Write down three main differences between flexgrid control and dbgrid control.
3. Difference between Recordset and Resultsets.
4. What is the advantages of disconnected recordsets?
5. How many data-bound controls are found in ADO data control?
6. What is the use of data environment and the ADO control in VB controls?

## **Answers for Self Assessment Questions**

1. (d)                      2. (a)                      3. (b)                      4. (c)                      5. (d)

## **12.9 Further Readings**



G. Cornell, "Visual Basic 6", Tata McGraw-Hill, 1998.

Deitel & Deitel, T.R. Nieto, "Visual Basic 6, How to program" Prentice Hall of India, 1999.



<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 13: Working With Reports**

### **CONTENTS**

Objectives

Introduction

13.1 Data Environment

13.1.1 Data Environment Contents

13.1.2 Drag and Drop Scenarios in the Data Environment

13.1.3 The Data Environment Object Model

13.2 Accessing Data

13.2.1 Middle Tier Components and Microsoft Transaction Server

13.3 ActiveX Data Objects (ADO)

13.4 Data Sources and Data Controls

13.5 Dynamic Data Binding

13.5.1 Presenting Data to the End User

13.6 Data Formatting and Data Validation

13.6.1 Language Features

13.6.2 DHTML and Data Access

13.7 Creating Various Type Data

13.8 Creating a Data Environment Report Using Grid

13.8.1 Creating a Data Report

13.8.2 Adding Data Report

13.8.3 Accessing the Data Report

13.9 Creating a Data Environment Report Using Data Bound Control

13.10 Summary

13.11 Keywords

13.12 Self Assessment Questions

13.13 Review Questions

13.14 Further Readings

### **Objectives**

*After studying this unit, you will be able to:*

- Discuss data environment
- Explain accessing data
- Explain activex data objects (ADO)



- Discuss data source and data control
- Understand dynamic data binding
- Discuss data formatting and data validation
- Explain creating various type of data
- Discuss data environment report using grid
- Explain data environment report using data bound control

## **Introduction**

Visual Basic 6 provides you with a data report designer to create your report, it is somewhat similar to data report designer in Microsoft Access. The data report designer has its own set of controls which allow you to customize your report seamlessly.

### **13.1 Data Environment**

The data environment is a repository in your Visual InterDev Web project for the information required in server script to connect and manipulate data in databases. It provides a standard interface for creating re-usable data-related objects and for placing them on Web pages.



Notes

The data environment is available on the server. If you are designing a Web application that uses client access to data (using Internet Explorer 4.0 DHTML), the data environment is available at design time, but not used at run time.

The data environment also provides an object you can reference in script, allowing you to access and manage database objects such as tables, views, stored procedures, and SQL commands programmatically. It provides an easy-to-use wrapper around ActiveX Data Objects (ADO), making these objects more accessible and easier to work with in Visual InterDev.

To understand the data environment, you must understand these concepts:

- Data Environment Contents
- Drag and Drop Scenarios in the Data Environment
- The Data Environment Object Model

#### **13.1.1 Data Environment Contents**

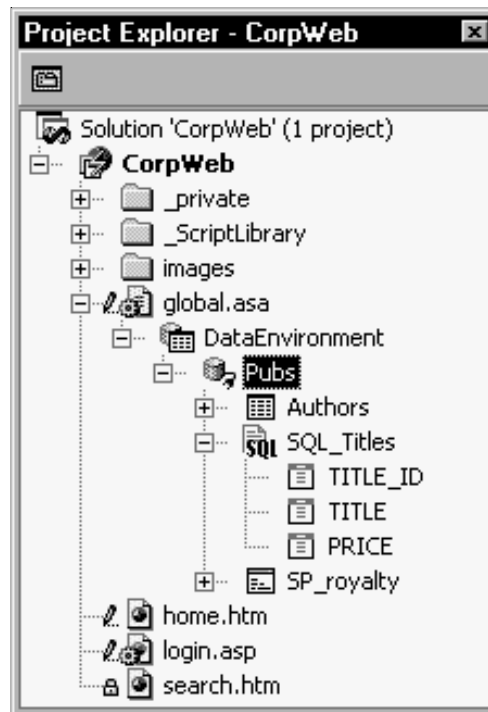
The primary component of the data environment is a data connection, which includes the information required to connect to one database with a specific user name. For example, your data environment might include a connection that links your application to the Pubs database on a SQL Server under the user names Admin (at design time) and Guest (at run time). If your application requires access to multiple databases, you can add multiple data connections to your data environment.

Within each data connection, you can add one or more data commands (command objects), which define a set of data to work with. Command objects can reference a database object such as a table, query, view, synonym, stored procedure, or SQL statement. For example, you might create a command object that references the Authors table so you can display the contents of that table on a Web page. You could also define additional command objects to reference queries and stored procedures you can call to display and update data in other tables.

Command objects are accessible to any page in your application. They therefore become reusable objects. If the underlying database changes, you can make a single change to the command object, and all Web pages that reference the command object will continue to work properly.

Each command object is a node that contains additional information relating to that command object. For example, a command object that references a table contains a list of columns in that table. A command object that references a stored procedure can contain a list of the columns returned by the procedure, or it can contain a list of the procedure's parameters. For information on adding Command objects, see Getting Records and Visual InterDev creates a data environment for your project the first time you define a data connection. As soon as you add the connection, Visual InterDev creates the Data Environment folder and adds it as a node under the Global.asa file. The data connection you added is displayed in the Data Environment node.

As you create additional data connections, they are added to the Data Environment node.



**Figure 13.1:** A Web Project Showing the Data Environment and Data Commands.



Notes

You can only have one Data Environment node (and one data environment) in a Visual InterDev project.

### 13.1.2 Drag and Drop Scenarios in the Data Environment

An important feature of the data environment is that you can drag objects to and from it to simplify the process of adding database access to your application. To create new commands, you can drag database objects from the Data View window to the data environment.

In addition, you can create data-bound controls on a page by dragging commands and database fields from the data environment to your page.

The following table summarizes how you can use drag and drop with the data environment?

| Drag                           | From             | Drop on                        | To                                                                                                                                                       |
|--------------------------------|------------------|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Database object                | Data View        | Connection in data environment | Create a command object for the database object you dragged. For example, dragging a table creates a command object whose Database Object type is table. |
| Command object or Field object | Data environment | Web page                       | Create a data-bound control.                                                                                                                             |
| Database object                | Data View        | Web page                       | Not allowed. Drag objects from the data environment instead.                                                                                             |

### 13.1.3 The Data Environment Object Model

The data environment supports its own object model, which you can use when writing script to manipulate the data you want to display on your Web page. The data environment object model is based on the ActiveX Data Objects (ADO) object model, but is simpler to use.

In ADO, the main objects in the data environment object model are the Connection object, Command object, Record set object, Field objects, and Parameter objects. Each of these ADO objects has its own properties and methods.

The data environment abstracts this object model to make it simpler to use. The data environment itself is an object that can be used in script and that contains these ADO objects. Within the data environment object, command objects are exposed as methods in script. You can call a command method to execute it and return the record set referenced by the command or to execute its SQL command or stored procedure.

Each of the data environment objects also has properties you can set in the property page for that object or directly in script.

## 13.2 Accessing Data

The Figure 13.2 is a roadmap of data access technologies found in Visual Basic. The Figure 13.2 features “hot” zones, which you can click to find out more information about any particular set of data, access tools or technologies.

Using Visual Basic 6.0 you can create components that encapsulate every step in a data access system. Beginning with the data source, Microsoft Visual Data Tools (accessible through the Data View window) give you the ability to view and manipulate tables, views, stored procedures, and database schemas on SQL Server and Oracle systems.

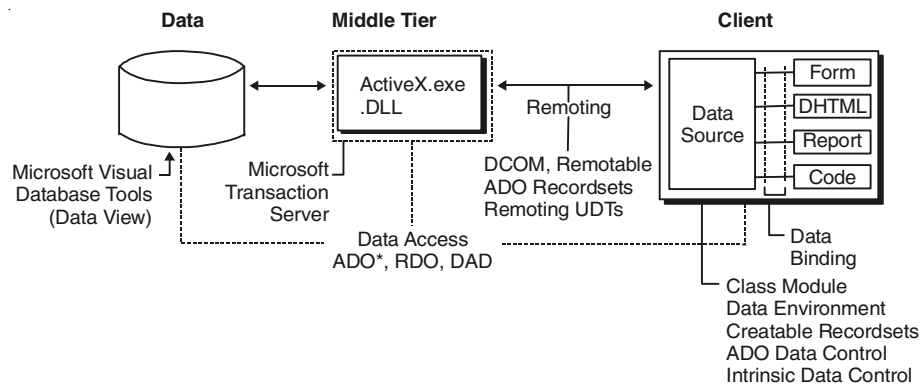


Figure 13.2: Microsoft Visual Data Tools.

### 13.2.1 Middle Tier Components and Microsoft Transaction Server

The power of Visual Basic is also leveraged to create the middle tier components in your application, as you make your own ActiveX DLLs and EXEs. Visual Basic now includes enhancements that tailor applications to work with Microsoft Transaction Server.

## 13.3 ActiveX Data Objects (ADO)

The bridge between the data providers and data consumers is through data sources created using Microsoft ActiveX Data Objects (ADO), which is the primary method in Visual Basic to access data in any data source, both relational and non-relational. For backward compatibility and project maintenance, Remote Data Objects (RDO) and Data Access Objects (DAO) are still supported.

## 13.4 Data Sources and Data Controls

A graphical designer that allows you to quickly create ADO Connections and Commands to access your data. The Data Environment designer provides a dynamic programmatic interface to the data access objects in your project. In addition, the Data Environment provides advanced data shaping services—the ability to create hierarchies of related data, aggregates, and automatic groupings, all without code.

The new ADO Data control is similar to the intrinsic data control and Remote Data control, except that it uses ADO to access data. You can now use an ADO Recordset as a data source for your controls and objects in Visual Basic.



Did u know?

In Visual Basic you can now create your own data sources either as user controls or classes, to encapsulate business rules or proprietary data structures. The class module now features the `DataSourceBehavior` property and the `GetDataMember` event, which allow you to configure a class as a data source.

## **13.5 Dynamic Data Binding**

The ability to dynamically bind a data source to a data consumer is now possible in Visual Basic. At run time, you can now set the DataSource property of a data consumer (such as the DataGridcontrol) to a data source (such as the ADO Data control). This capability, unavailable in previous versions of Visual Basic, allows you to create applications, which can access a multitude of data sources.

### **13.5.1 Presenting Data to the End User**

Visual Basic offers a variety of rich ways to present data to your end users. ADO/OLE DB-based versions of all the data bound controls are included in Visual Basic:

- The DataList and DataCombo controls are the ADO/OLE DB equivalents of DBList and DBCombo controls.
- The DataGrid is the successor to DBGrid.
- The Chart control is now data bound.
- A new version of the FlexGrid control, called the Hierarchical FlexGrid, supports the hierarchical abilities of the Data Environment.
- The new DataRepeater control functions as a scrolling container of data bound user controls where each control views a single record.

The Data Report is a new ActiveX designer that creates reports from any data source, including the Data Environment. With the Data Report designer, formatted reports can be viewed online, printed, or exported to text or HTML pages.

## **13.6 Data Formatting and Data Validation**

The new Data Format object allows you to display data with custom formatting, but write it back to the database in the native format. For example, you can now display dates in the format appropriate to a country, while the actual data is stored in a date format. Data is formatted coming out of the source, and unformatted going back in. You can also do custom formatting and perform additional checks using the Format and Unformat events.

Data validation is also enhanced using the CausesValidation property with the Validate event. By setting the CausesValidation property to True, the Validate event for the previous control in the tab order will occur. Thus, by programming the Validate event, you can prevent a control from losing focus until the information it contains has been validated.

### **13.6.1 Language Features**

New data-related enhancements to the Visual Basic language include the ability to pass User-defined Types (UDTs) and arrays across processes. You can now define a UDT and pass it as a parameter to another process, such as an ActiveX EXE or DLL.

### **13.6.2 DHTML and Data Access**

Using Visual Basic, you can create complete web applications for data access. All of the data tools and technologies can also be used in DHTML pages, and on web server (IIS) applications.

### 13.7 Creating Various Type Data

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

There are two steps to creating a data report. First, we need to create a Data Environment. This is designed within Visual Basic and is used to tell the data report what is in the database. Second, we create the Data Report itself. This, too, is done within Visual Basic. The Data Environment and Data Report files then become part of the Visual Basic project developed as a database management system.

The Visual Basic 6.0 data report capabilities are vast and using them is a detailed process. The use of these capabilities is best demonstrated by example. We will look at the rudiments of report creation by building a tabular report for our phone database.



#### *Example: Phone Directory—Building a Data Report*

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

### 13.8 Creating a Data Environment Report Using Grid

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

Command Name—PhoneList

Connection—Phone

DataBase Object—Table

ObjectName—PhoneList

5. Click **OK**. All this was needed just to connect the environment to our database.
6. Display the properties window and give the data environment a name property of denPhone. Click **File** and **Save** denPhone As. Save the environment in an appropriate

folder. We will eventually add this file to our phone database management system. At this point, my data environment window looks like this (I expanded the PhoneList tab by clicking the + sign):

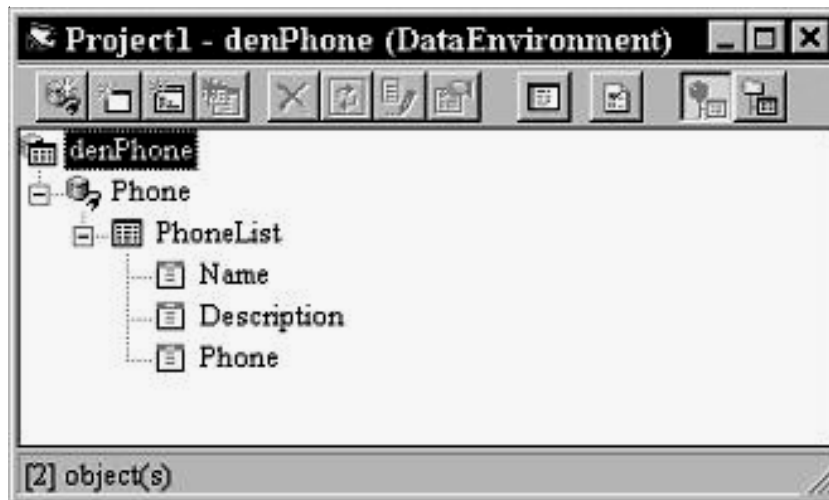


Figure 13.3: Data Environment Window.

### 13.8.1 Creating a Data Report

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

1. On the Project menu, click Add Data Report and one will be added to your project. If this item is not on the menu, click Components. Click the Designers tab, and choose Data Report and click OK to add the designer to your menu.
2. Set the following properties for the report:  
 Name—rptPhone  
 Caption—Phone Directory  
 DataSource—denPhone (your phone data environment—choose, don't type)  
 DataMember—PhoneList (the table name—choose don't type)
3. Right-click the **Data Report** and click **Retrieve Structure**. This establishes a report format based on the **Data Environment**.
4. Note there are five sections to the data report: a Report Header, a Page Header, a Detail section, a Page Footer, and a Report Footer. The headers and footers contain information you want printed in the report and on each page. To place information in one of these regions, right-click the selected region, click Add Control, then choose the control you wish to place. These controls are called data report controls and properties are established just like you do for usual controls. Try adding some headers.
5. The Detail section is used to layout the information you want printed for each record in your database. We will place two field listings (Name, Phone) there. Click on the Name tab in the Data Environment window and drag it to the Detail section of the Data Report.

Two items should appear: a text box Name and a text box Name (PhoneList). The first text box is heading information. Move this text box into the Page Header section. The second text box is the actual value for Name from the PhoneList table. Line this text box up under the Name header. Now, drag the Phone tab from the Data Environment to the Data Report. Adjust the text boxes in the same manner. Our data report will have page headers Name and Phone. Under these headers, these fields for each record in our database will be displayed. When done, the form should look something like this:



Figure 13.4: Data Report.

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

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

### 13.8.2 Adding Data Report

**Step 1:** Start Visual Basic as a Standard EXE project. From the Project menu in the VBE, select Add Data Report in the dropdown menu. Now, you will be presented with the data report environment, as shown in Figure 13.5. The data report environment contains 6 controls, they are RptTextBox, RptLine, RptFunction, RptLabel, RptImage and RptShape.

You can customize your report here by adding a title to the page header using the report label RptLabel. Simply drag and draw the RptLabel control on the data report designer window and use the Caption property to change the text that should be displayed. You can also add graphics to the report using the RptImage control.



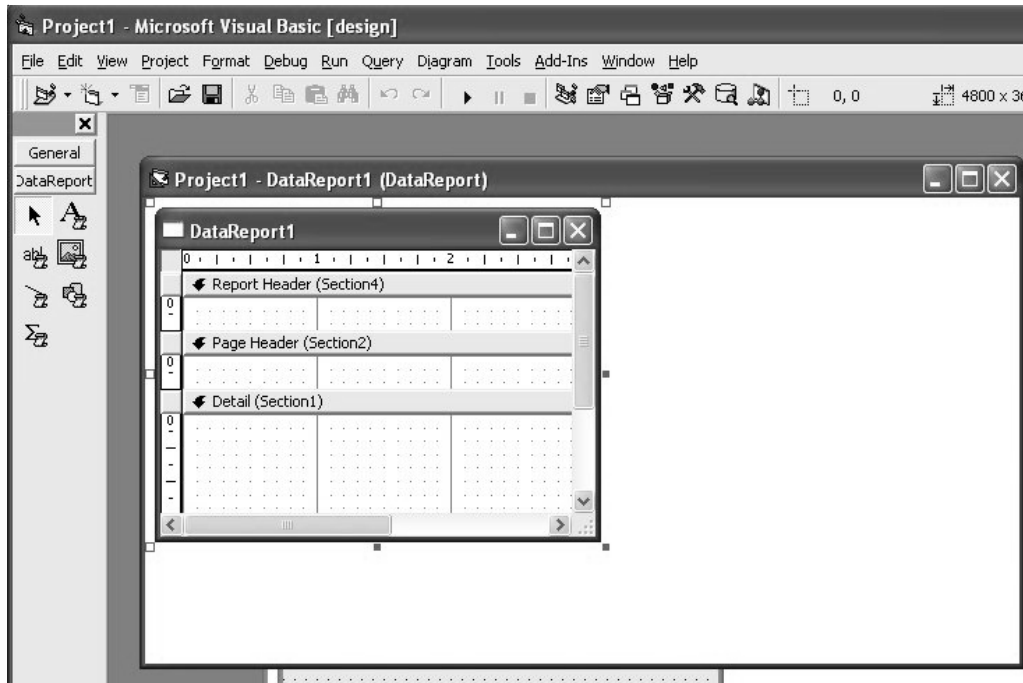


Figure 13.5: The Data Report Environment.

**Step 2:** Connecting the report to database using Data Environment Designer Click the Project menu, then select Data Environment from the drop-down menu. The default data environment will appear, as shown in Figure 13.6.

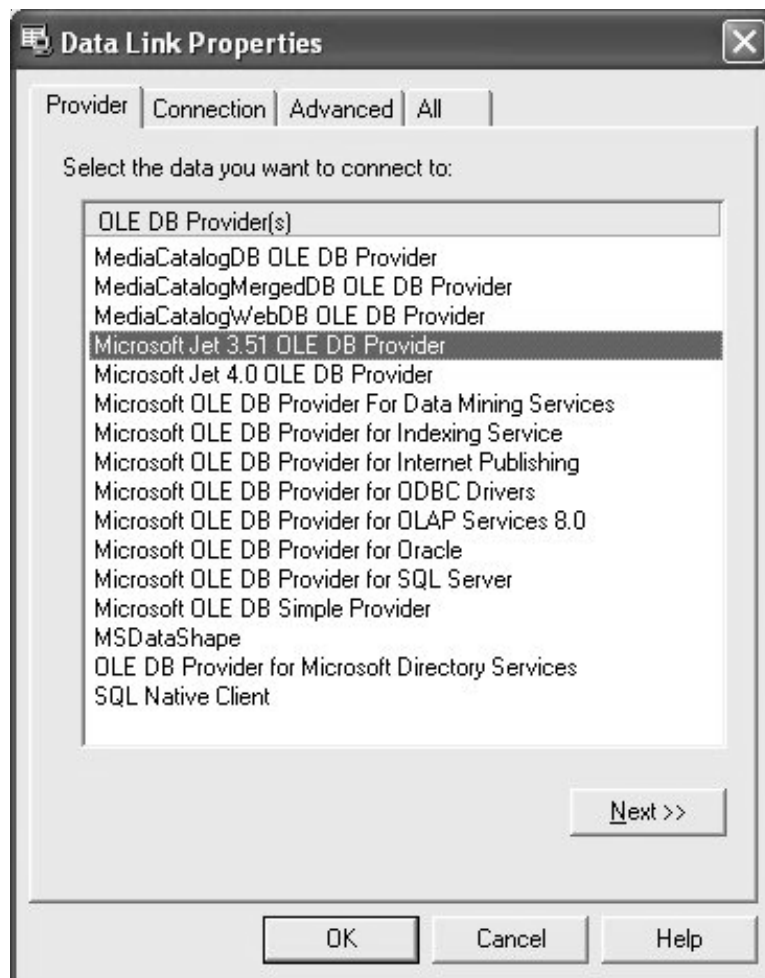


Figure 13.6: Data Environment.



Connection string must be correct in the connection string option.

Now, to connect to the database, right-click connection1 and select Microsoft Jet 3.51 OLE DB Provider (as we are using MS Access database) from the Data Link Properties dialog (as shown in Figure 13.7), then click next.



**Figure 13.7:** Data Link Properties Dialog.

Now, we need to connect to the database by selecting a database file from your hard disk. For demonstration purpose, we will use the database BIBLIO.MDB that comes with Visual Basic, as shown in Figure 9.8. The path to this database file is C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB. This path varies from computers to computers, depending on where you install the file. After selecting the file, you need to test the connection by clicking the Test Connection button at the right bottom of the Data Link Properties dialog. If the connection is successful, a message that says 'Test Connection Succeeded' will appear. Click the OK button on the message box to return to the data environment. Now rename connection1 to any name by right-clicking it. For example, you can change it to MyConnection. You may also change the name of DataEnvironment1 to MyDataEnvironment using the Properties window.

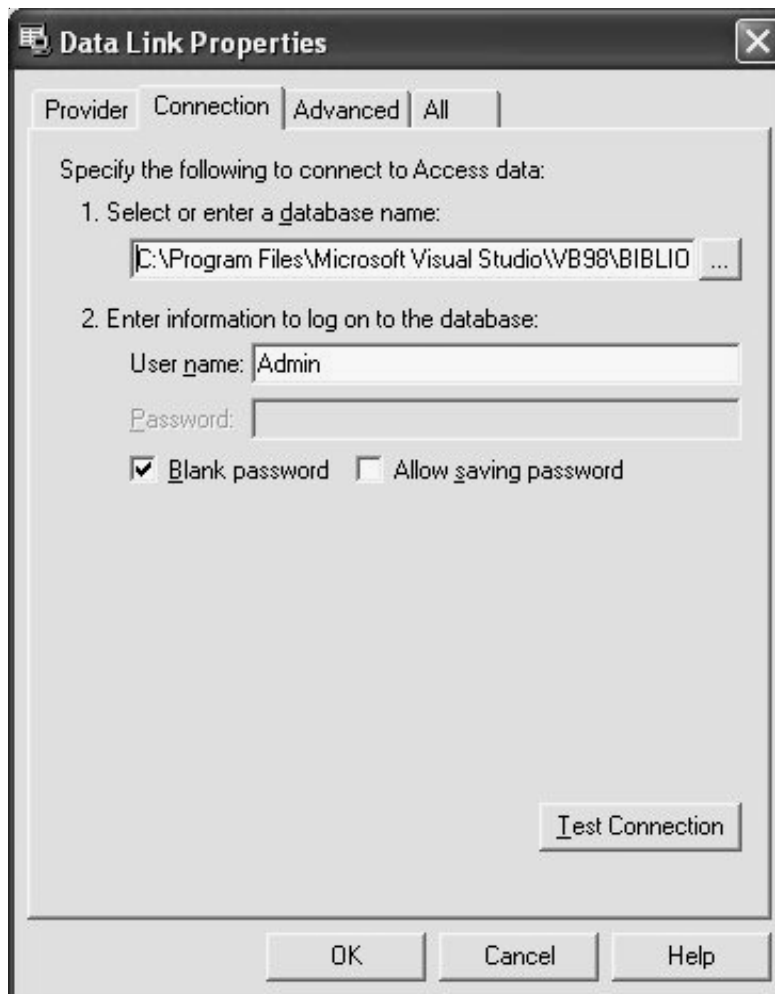


Figure 13.8: Data Link Property.

**Step 3:** Retrieving Information from the Database.

In order to use the database in your report, you need to create query to retrieve the information from the database. Here , we will use SQL command to create the query. First of all, right click on MyConnection to add a command to the data environment. The default command is Command1, you can rename it as MyCommand, as shown in Figure 13.9.



Figure 13.9: MyCommand.

In order to use SQL command, right-click MyCommand and you can see its properties dialog. At the General tab, select SQL statement and key in the following SQL statement:

```
SELECT Au_ID, Author
FROM Authors ORDER BY Author
```

This command is to select all the fields from the Authors table in the Biblio.Mdb database. The command ORDER BY Author is to arrange the list in ascending order according to the Authors' Names.

Now, you need to customize a few properties of your data report so that it can connect to the database. The first property to set is the DataSource, set it to MyDataEnvironment. Next, you need to set the DataMember property to MyCommand, as shown in Figure 13.10.

To add data to your report, you need to drag the fields from MyCommand in MyDataEnvironment into MyDataReport, as shown in Figure 13.11. Visual Basic 6 will automatically draw a RptTextBox, along with a RptLabel control for each field on the report. You can customize the look of the labels as well as the TextBoxes from the properties window of MyDataReport.

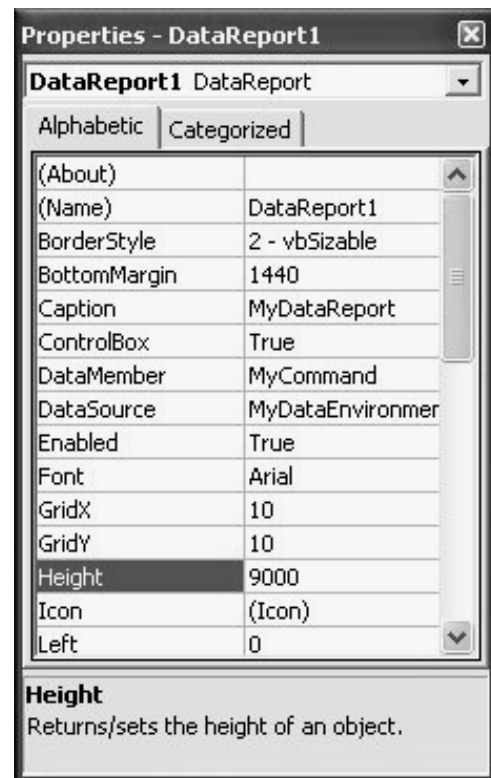


Figure 13.10: Properties.

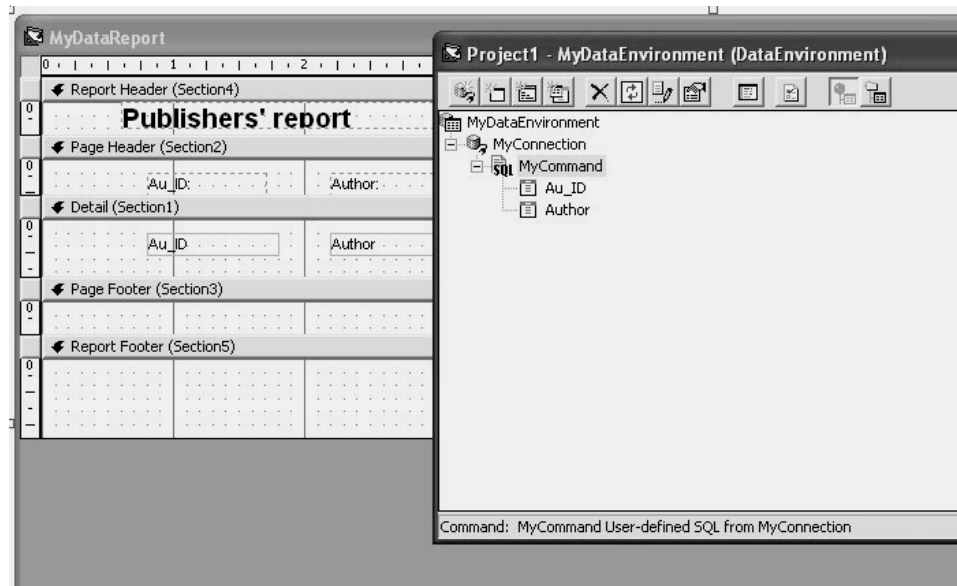


Figure 13.11: Data Report.

The Final step is to set MydataReport as the Startup form from the Project menu, then run the program. You will see your report as shown in Figure 13.12. You can print out your report.

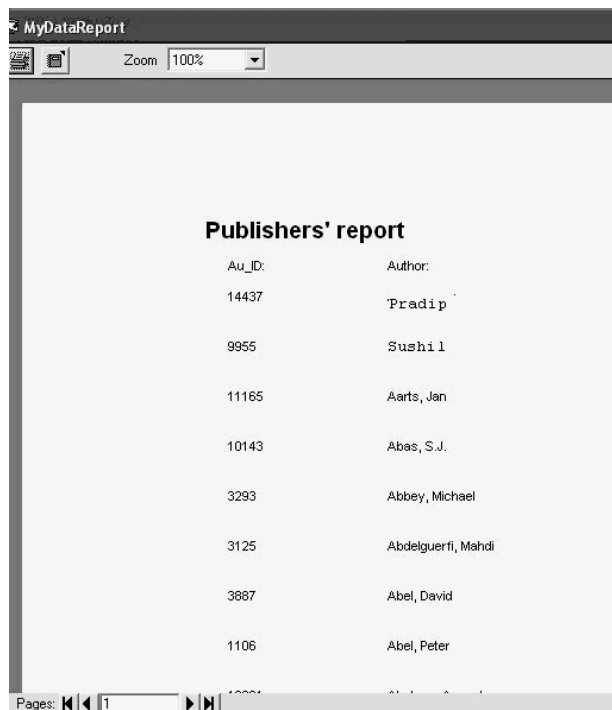


Figure 13.12: The Final Report.



Make a table and show the data in data grid view.

### 13.8.3 Accessing the Data Report

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

```
Private Sub cmdReport_Click()
 rptPhone.Show
End Sub
```

4. This uses the Show method to display the data report.
5. Save the application and run it. Click the Show Report button and this should appear:



Figure 13.13: Report.

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

## 13.9 Creating a Data Environment Report Using Data Bound Control

Steps to add data report and how to setup its DataSource, DataMember and DataField to display the record that you like.

1. Add Data Report to your project

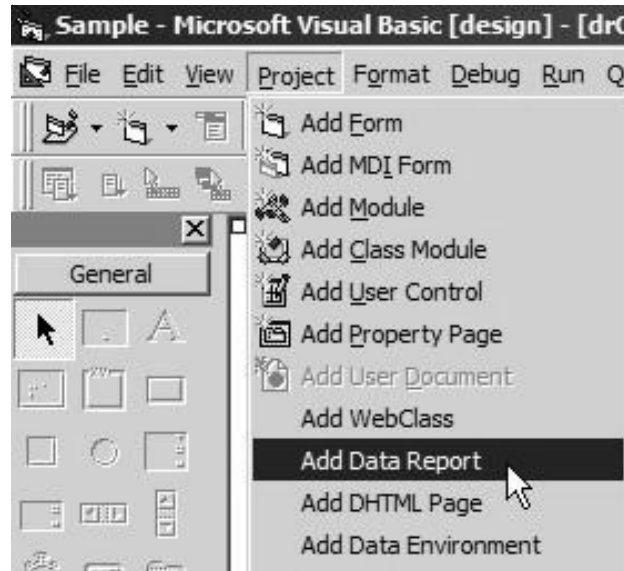


Figure 13.14: Adding Data Report.

2. Set DataSource and DataMember to your Data Report

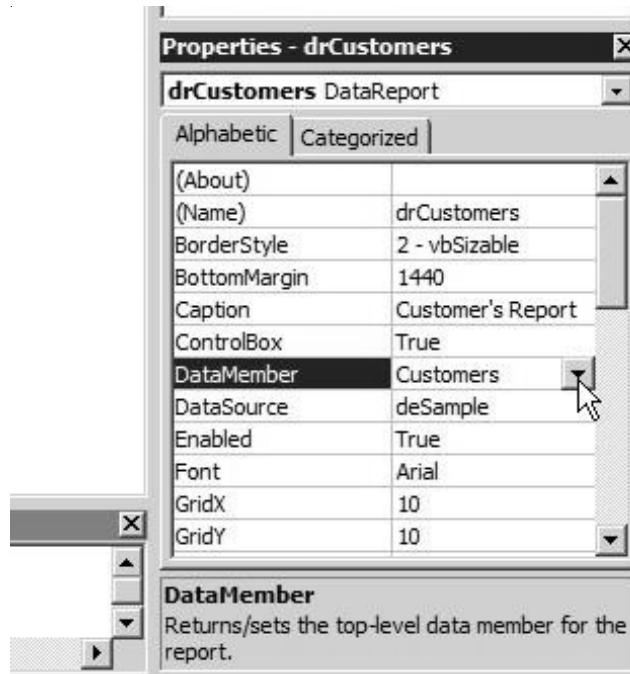


Figure 13.15: Managing Property.

3. Add a TextBox to the detail section of your Data Report

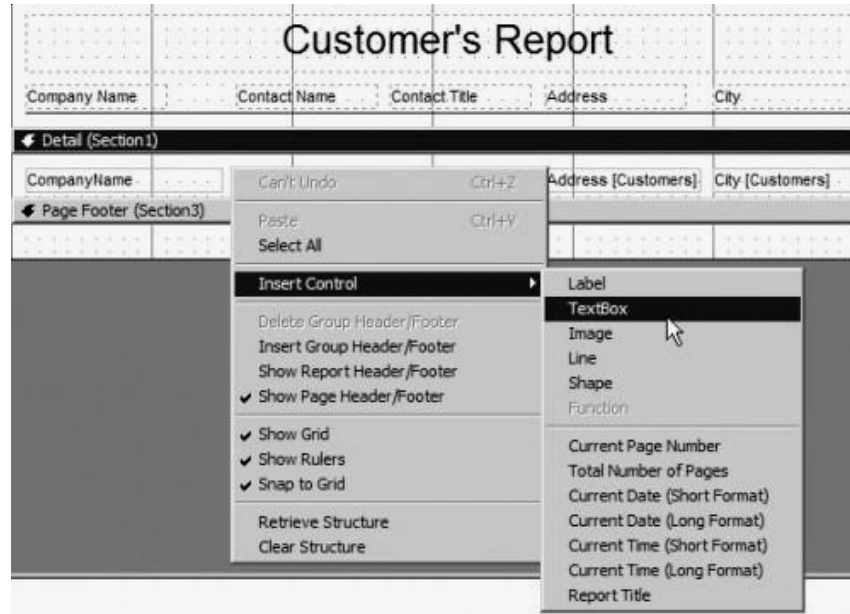


Figure 13.16: Report View.

4. Set the DataMember and DataField to the TextBox

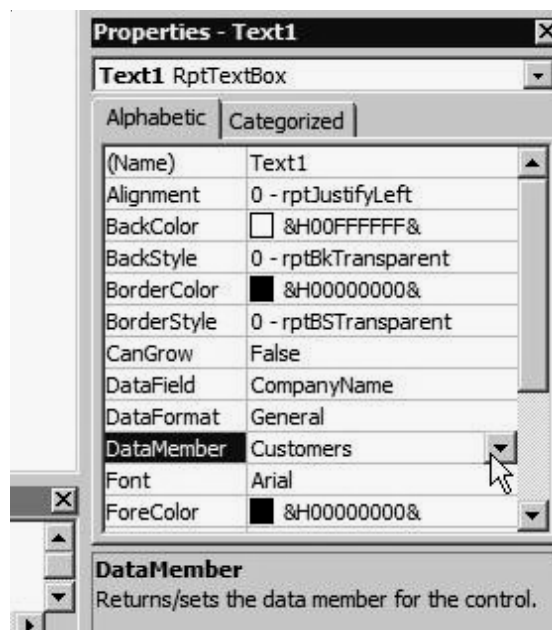


Figure 13.17: Property of Text.



5. Repeat these steps to add more TextBox and bound it to the DataMember and DataField based on the field you select.

Additionally we have created a button called CmdPrint to open newly created Data Report. Within the click event of this button put the code below:



#### Case Study

##### Create a Data-Bound Report

You can create a data-bound report using the same drag-and-drop technique you used to create data-bound forms. Simply create a Command object based on a table or query, then drag it onto a blank report in the Data Report designer.

In this topic, we will create a report that displays orders for French customers in the Northwind Traders sample database.

To create a data-bound report

1. Create a data environment Command object.
2. Open a Data Report designer.
3. Drag the Command object from the Data Environment designer to the Data Report designer.
4. Set data properties for the report.

##### Questions

1. How to create Data environment Command object?
2. What is data Report Designer?

### 13.10 Summary

- Data environment is a repository in our Visual InterDex.
- The activeX data object (ADO), which is the primary method in visual basic to access data in any data source, both relational and non-relational.
- In visual basic we can create our own data source either as user controls or classes to encapsulate business rules or proprietary data structure.
- Data formatting objects allow to display data with custom formatting.
- The select \* command is to select all the fields from the table in the database.

### 13.11 Keywords

**ActiveX Data Objects (ADO):** An ADO is a set of Component Object Model (COM) objects for accessing data sources.

**Data control:** A *Data control* is used as a mechanism for binding controls to a database using DAO.

**Data environment:** The *data environment* provides a standard interface for creating re-usable data-related objects and for placing them on Web pages.

**Data formatting:** The *Data format* object allows to display data with custom formatting, but write it back to the database in the native format.

**Data source:** A *data source* is a readily accessible object that provides data to any data consumer (any class or control that can be bound to a source of external data).

**Data validation:** A *data validation* is also enhanced using the CausesValidation property with the Validate event.

**Headers and footers:** It contains information we want printed in the report and on each page.



*Lab Exercise*

1. Write the program for add data from a ADO.
2. How to make a table in Data base?

### **13.12 Self Assessment Questions**

1. The data environment is a repository in your Visual InterDev Web  
(a) True (b) False
2. The primary component of the data environment is a data connection  
(a) True (b) False
3. The data environment supports  
(a) Object modal (b) Subject model  
(c) Create modal (d) None of the above
4. The power of Visual Basic is also leveraged to create the \_\_\_\_\_ in your application.  
(a) Single tier components (b) Middle tier components  
(c) Two tier components (d) None of the above
5. On the client side, several new data sources are available  
(a) True (b) False
6. The ability to dynamically bind a data source to a data consumer is  
(a) True (b) False
7. Data validation is also enhanced using the \_\_\_\_\_ property  
(a) Validate event (b) Causes Validation  
(c) Both of above (d) None of these

8. Once you have gone to all the trouble of \_\_\_\_\_ a database
  - (a) Developing
  - (b) Managing
  - (c) Developing and managing
  - (d) None of the above
9. There are two steps to creating a
  - (a) Data report
  - (b) Data Environment
  - (c) Data adopter
  - (d) None of these
10. In order to use the database in your report, you need to create query to retrieve the Information from the application
  - (a) True
  - (b) False

### **13.13 Review Questions**

1. What is RDO?
2. What are the two main objects that you can place on the surface of a Data Environment?
3. Which ADO object is always associated with a cursor?
4. Describe some of the basic characteristics of Date Environment.
5. What are the primary components of the data environment?
6. Name the four different cursor and locking types in ADO and describe them briefly.
7. Under the ADO Command Object, what collection is responsible for input to stored procedures?
8. How to create a Data Environment Report Using Grid?
9. How to setup DataSource, DataMember and DataField to display?
10. Which property of a Recordset object should you check to see if the preceding call to the Find method was successful?
11. How could you prevent one of ADO's Will event procedures from running multiple times during an application session?
12. What is the purpose of the ADO Errors collection?
13. You've been asked to build a customer phone list report. The phone list will be populated from a table named CUSTLIST, found in the SALES98 database on a SQL server named FARGO. In order to keep development time to a minimum, you are asked to use the Data Environment Designer. This report will be the only purpose of this application. After selecting a new Data Project template, what steps should you take to build this report?
14. How do you dynamically bind a text box control to an ADO Recordset object?
15. How can you place a DataGrid on a form that is automatically bound to the Recordset of a Data Environment's Command object? (Select all that apply.)

### **Answers for Self Assessment Questions**

- |        |        |        |        |         |
|--------|--------|--------|--------|---------|
| 1. (a) | 2. (a) | 3. (a) | 4. (b) | 5. (a)  |
| 6. (a) | 7. (b) | 8. (c) | 9. (a) | 10. (b) |

### **13.14 Further Readings**



*Books*

G. Cornell, "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net".



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>

## **Unit 14: Building Small Application**

### **CONTENTS**

Objectives

Introduction

14.1 Creating Visual Basic Project

14.2 Creating Visual Basic Form

14.3 Sample Project Specifications

14.4 Building a Calculator with Visual Basic

14.4.1 Instructions

14.5 Summary

14.6 Keywords

14.7 Self Assessment Questions

14.8 Review Questions

14.9 Further Readings

### **Objectives**

*After studying this unit, you will be able to:*

- Discuss creating visual basic projects and forms
- Explain sample project specification
- Explain building a calculator with visual basic and instructions

### **Introduction**

Microsoft Visual Basic is a programming language based on the BASIC language, originally developed to make programming easier to learn. Visual Basic takes the familiar commands of BASIC and adds object-oriented tools and interfaces for designing WYSIWYG-like Windows applications and web controls, among many other enhancements.

#### **14.1 Creating Visual Basic Project**

When starting a new VB project, make sure it is a Standard EXE. The more advanced types will be covered later. The first object in a new Visual Basic Project is the Form. We can set a few basic properties for the form. Those properties will apply to all the objects (controls) we place onto the form. The Visual Basic Project is created the first time we do a Save. Make sure to pick a meaningful name and save it into a folder that we'll be able to find later. It's amazing how many projects get lost in the file structure because people can't remember where they put them and how they were named.

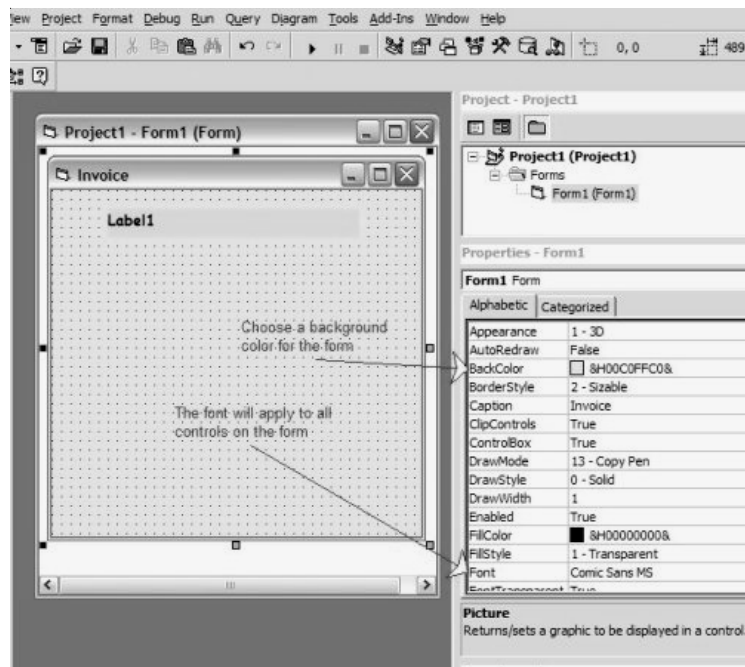


Fig. 14.1

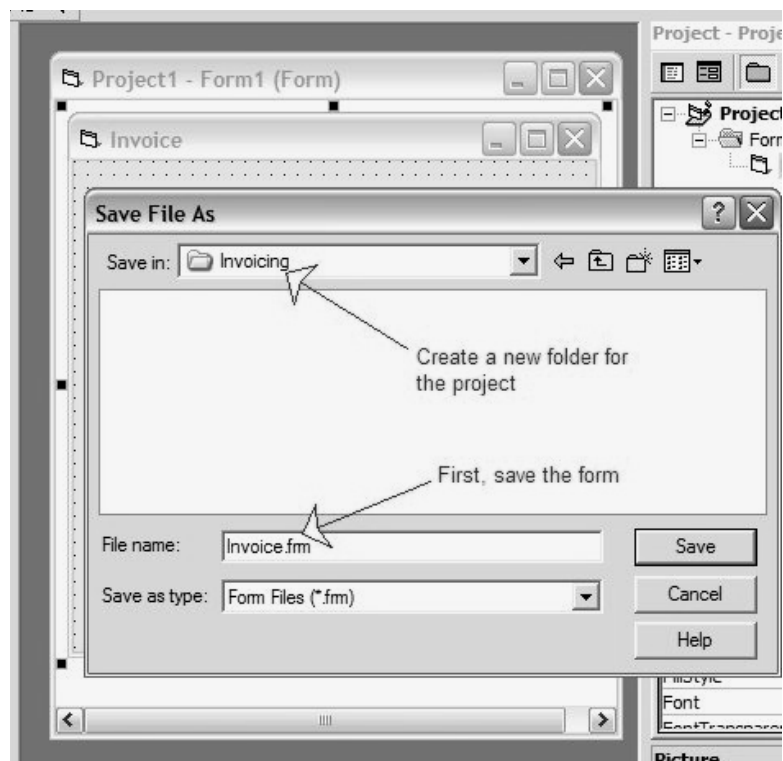


Fig. 14.2



Did u know?

#### To create a project for your program:

From the Windows **Start** menu, choose Microsoft Visual Basic 2005 Express Edition. The “Welcome to Visual Basic Express” screen appears. This is the interface for Visual Basic 2005 Express Edition, also known as the integrated development environment or IDE.

1. On the **File** menu, click **New Project**.

The **New Project** dialog box opens.

2. Select **Windows Application** and click **OK**.

A new form displays in the IDE, and the necessary files for project are added to the **Solution Explorer** window. If this is the first **Windows Application** project that we have created, it is named “WindowsApplication1”.

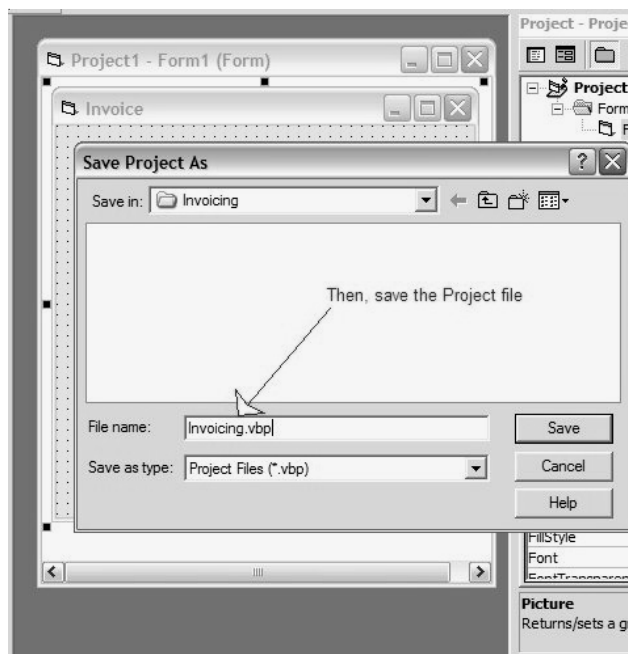


Fig. 14.3



Task

How to change back colour in visual basic 6.0, explain with steps?

## 14.2 Creating Visual Basic Form

It is recommended however that we create our own Project to become more familiar with all the details of form creation and maintenance. As mentioned in the previous tutorial, we must first assign a color and font to the form and then name it and save it. Then we will create some basic controls. Visual Basic Controls are objects that determine what the form does. Labels, Buttons, Textboxes are all controls. We get the controls from the Toolbox and drag them onto the form.

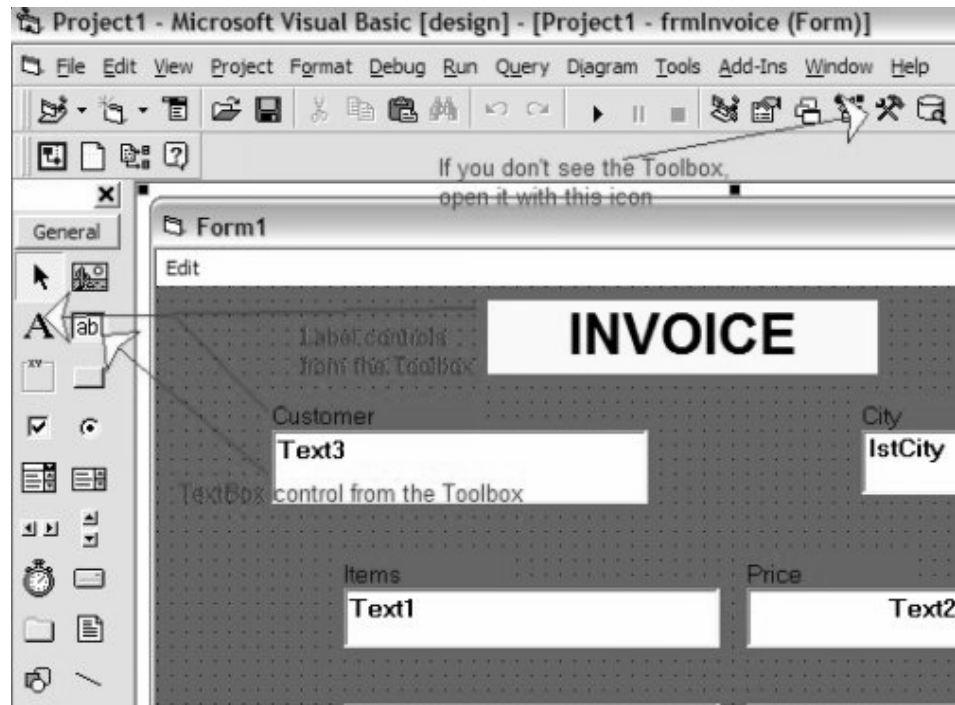


Fig. 14.4

### **14.3 Sample Project Specifications**

We must build a form so that the user at the Order Desk can input a customer's purchases. The user will input the customer's name and will select the customer's city from a list of 5 cities that will be shown to him. If the customer is from 'Capital City' he must pay 8% tax on his purchases otherwise, there is no tax. The user will then input from 1 to 3 items that the customer purchased along with the price of each item. The maximum price for an item is 50. Anything over that is 'Too big'. If there is a price entered for an item there must be a description also. If an item is being returned instead of purchased, the price will be negative and a red flag with the word 'Credit' will appear next to the price. When the user hits the 'Calculate' button, the subtotal, the tax payable and the final total will appear. There must be the usual buttons to clear the form and to exit when finished. There is also a need for a menu function to be able to cut, copy and paste text from different parts of the form. The first thing to do is to create all the required controls. However, because the items and the prices are a special case, don't do them yet. This form should look something like this:

### **14.4 Building a Calculator with Visual Basic**

Microsoft Visual Basic is a programming language based on the BASIC language, originally developed to make programming easier to learn. Visual Basic takes the familiar commands of BASIC and adds object-oriented tools and interfaces for designing WYSIWYG-like Windows applications and web controls, among many other enhancements. One relatively simple learning project for Visual Basic is the creation of a Windows calculator.





Fig. 14.5

### 14.4.1 Instructions

1:

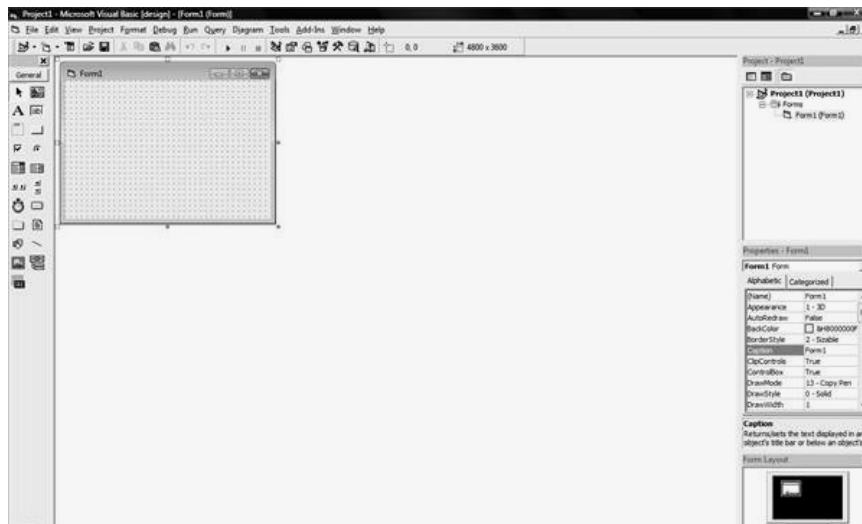


Fig. 14.6: New project with blank form (ALLD)

Open Visual Basic 6, and select “Standard EXE” from the new project menu. We will see a blank form appear on the screen.

2:

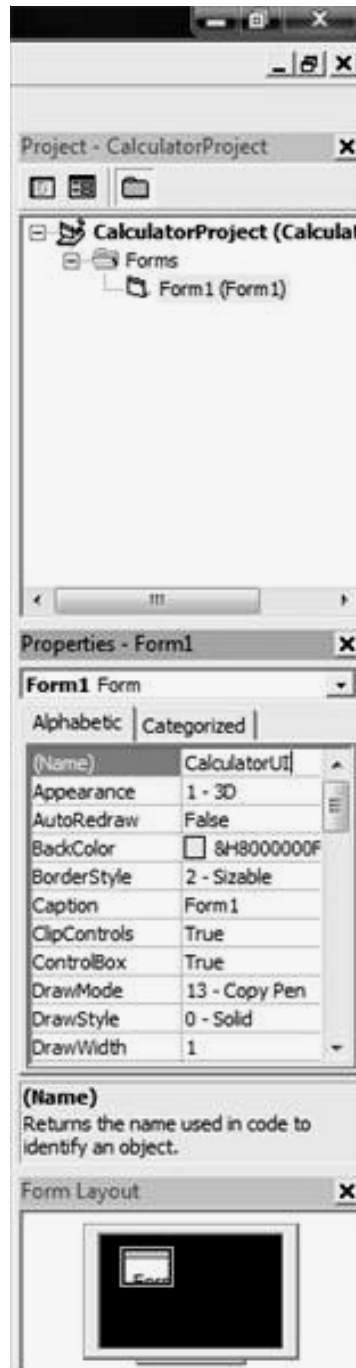


Fig. 14.7

Project listing and properties window (ALLD) rename project and form by clicking on “Project1” at the right hand side of the screen, in the project listing, and entering a new name in the “Name”

line of the Properties box, which should appear below the project listing by default. Press “Enter” to accept the new name. Do the same for form (a suggested form name is “CalculatorUI”), making sure to enter a similar name in the “Caption” property as well, which will change the text in the top bar of the form. Save the project in a new folder on computer.

3:

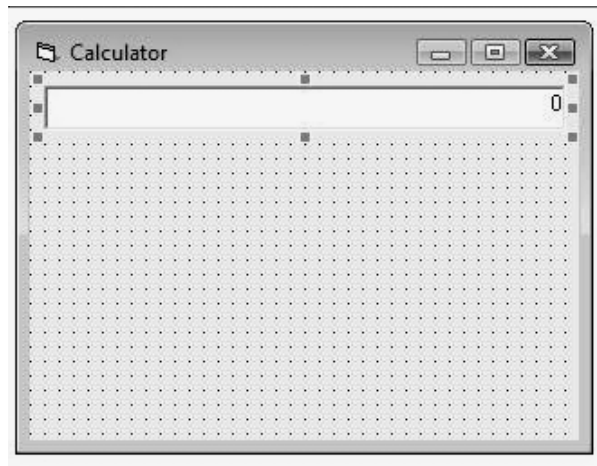


Fig. 14.8: Textbox creation on form (ALLD)

Add buttons and a text box to the form. First, add a text box, which will be where the numbers entered in the calculator appear, as well as the results of calculations. Do this by selecting the TextBox button from the toolbar at the left side of the screen, and then dragging with mouse the size and location desired for the TextBox. Once we placed the TextBox we can change the size and location by dragging it to another location of the form or by dragging the handles (the small squares) along the border of the TextBox. Be sure to change the following lines in the Properties window, with the TextBox selected: “(Name)” = tbResult, “Alignment” = 1 - Right Justify, “Data Format” = (click on the “.” button to select) Number, “Locked” = True, and “Text” = 0.

4:

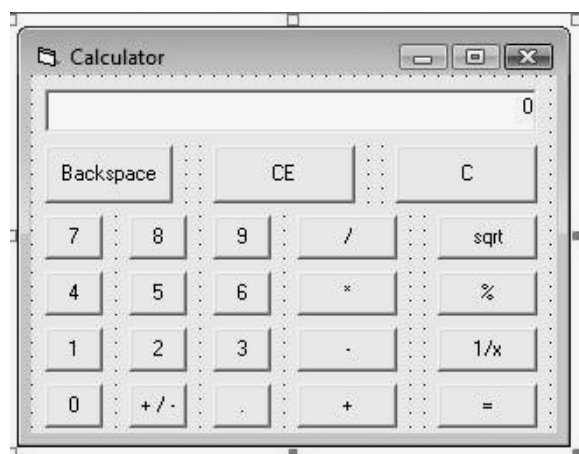


Fig. 14.9: Button layout for the calculator program (ALLD)

Select the CommandButton icon on the toolbar and create the first button the same way we created the TextBox to add buttons. For reference, use the Windows calculator in Standard view (Programs > Accessories > Calculator) as a basis for calculator layout, leaving out the “MC”, “MR”, “MS”, and “M+” buttons. On each button, change the following properties (using the “+” button as an example): “(Name)” = btnPlus, “Caption” = +. Do the same for the rest of the calculator buttons, and then save work. This form should now resemble the example shown here.

5: Add the code. Note that if buttons and textbox are not named the same as the code listed here expects, we will need to change the names to match buttons and textbox, or change buttons and textbox to match this code. First we need to create a few variables for processing calculator input:

```
Dim sLeft As String, sRight As String, sOperator As String
Dim iLeft As Double, iRight As Double, iResult As Double
Dim bLeft As Boolean
```

Each calculation consists of four parts: a number to the left of the operator (sLeft, iLeft), an operator (sOperator), a number to the right of the operator (sRight, iRight), and a result (iResult). In order to track whether the user is entering the left or right number, we need to create a boolean variable, bLeft. If bLeft is true, the left side of the calculation is being entered; if bLeft is false, the right side is being entered.

6: Initialize the bLeft variable. We do that by creating a Form\_Load subroutine, which we can either type as listed here or automatically create by double-clicking on any part of the form not covered by a button or textbox. Inside the function, we need to set bLeft to True, because the first number entered will be the left part.

```
Private Sub Form_Load()
 bLeft = True
End Sub
```

7:

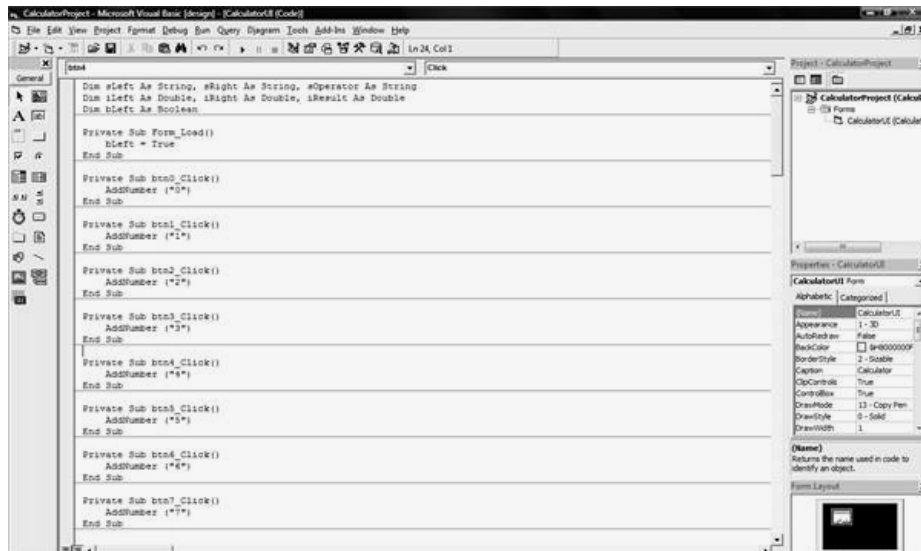


Fig. 14.10: Code listing after step 7 (ALLD)

Create a subroutine that will handle the clicking of any of the number buttons. We create this as a subroutine because we use identical code for each button, and using a subroutine means not having to repeat the same code ten times. Enter the following below the Form\_Load subroutine's End Sub line:

```
Private Sub AddNumber(sNumber As String)

If bLeft Then

sLeft = sLeft + sNumber

tbResult.Text = sLeft

Else

sRight = sRight + sNumber

tbResult.Text = sRight

End If

End Sub
```

As we can see, this function takes a string parameter, sNumber, which will contain the number the user has clicked on. If bLeft is true, this number is appended to the string that represents the number being entered, sLeft, and the textbox, tbResult, is updated to display the new number. If bLeft is false, the same operation is performed using sRight instead.

Finally, create a Click event function for each number that calls our AddNumber subroutine. We can do this easily by double-clicking each number button, which will create the subroutine structure. Then add the call to AddNumber, replacing the number in quotes with the number associated with the button for the zero button, code will look like this:

```
Private Sub btn0_Click()

AddNumber ("0")

End Sub
```

Likewise, for the one button, code will look like this

```
Private Sub btn1_Click()

AddNumber ("1")

End Sub
```

**8:** Handle the operators: plus, minus, times, and divide. We will do this like last step, creating a subroutine that is called in the Click events for the operator buttons. The subroutine will look like the following:

```
Private Sub AddOperator(sNewOperator As String)

If bLeft Then

sOperator = sNewOperator

bLeft = False

Else

btnEquals_Click
```

```
sOperator = sNewOperator
sRight = ""
bLeft = False
End If
End Sub
```

If bLeft is true, meaning the user has just entered the left part of the calculation, this subroutine sets the sOperator variable we created in step 5 to equal the operator entered, which is passed to AddOperator as the string sNewOperator. The second step is to set bLeft to False, because the entry of an operator means the user is done entering the left side of the equation. In order to handle entries that string multiple operators together, such as  $9 * 3 * 2 * 6$ , we need to also check whether bLeft is false, meaning the user has entered an operator where we were expecting an equals. First we call the Click event for the equals button (described in the next step), which does the calculation and sets tbResult to the result of what has already been entered. Then we clear sRight so the user can enter the next number, and set bLeft to False so the program knows we are entering the right hand side of the calculation next. Finally, add an AddOperator call to the Click event of each operator button, using the same method as we used in step 7 to create the Click events for the number buttons. Code for the plus button will look like this:

```
Private Sub btnPlus_Click()
AddOperator ("+")
End Sub
```

Likewise, the code for the minus button will look like this:

```
Private Sub btnMinus_Click()
AddOperator ("-")
End Sub
```

**9:** Create the Click event for the equals button, which is the most complex code in this program. Create the subroutine structure like we did for the other buttons, by double-clicking the equal's button on our form. Our subroutine will look like this when we've entered the code:

```
Private Sub btnEquals_Click()
If sLeft <> "" And sRight = "" And sOperator <> "" Then
sRight = sLeft
End If

If sLeft <> "" And sRight <> "" And sOperator <> "" Then
iLeft = sLeft
iRight = sRight
Select Case sOperator
Case "+"
iResult = iLeft + iRight
Case "-"
```

```
iResult = iLeft - iRight
Case "/"
iResult = iLeft / iRight
Case "*"
iResult = iLeft * iRight
End Select
tbResult.Text = iResult
sLeft = iResult
sRight = ""
bLeft = True
End If
End Sub
```

The first three lines of code check to see if both sides of the calculation have been entered along with an operator. If only the left side and an operator are entered, the value of the left side is copied to the right, so we can mimic the standard calculator behavior for handling an entry like  $9 * =$ , which multiplies 9 by itself to get a result of 81. The rest of the code will run only if left, right, and operator are entered, and starts out by copying the strings of numbers into our iLeft and iRight Double-typed variables, which can do the actual calculations. The Select Case statement allows us to run different code depending on which operator was entered, and performs the actual calculation, placing the result in iResult. Finally, we update the textbox with the result, copy the result into sLeft, reset sRight, and set bLeft = True. These last lines allow us to take the result of the calculation and use it to perform another calculation.

**10:** Handle the last three operation buttons: sqrt, %, and 1/x. For the Click event of the square root button, Code will look like this:

```
Private Sub btnSqrt_Click()
If sLeft <> "" Then
iLeft = sLeft
Else
iLeft = 0
End If
If sRight <> "" Then
iRight = sRight
Else
iRight = 0
End If
If bLeft Then
iLeft = Math.Sqrt(iLeft)
```

```
tbResult.Text = iLeft
Else
iRight = Math.Sqrt(iLeft)
tbResult.Text = iRight
End If
If iLeft <> 0 Then
sLeft = iLeft
Else
sLeft = ""
End If
If iRight <> 0 Then
sRight = iRight
Else
sRight = ""
End If
End Sub
```

The first 11 lines of code make sure that if we don't have a value entered for either side of the equation, we substitute zero instead of trying to copy an empty string into iLeft or iRight, which will generate an error. The middle lines perform the square root function on the current part of the calculation, either left or right. Finally, we reverse the checks we did in the beginning so that a zero is copied as an empty string back into sLeft and sRight. For the percent button, the code is similar, with one exception: the percent operation can only be performed if both left and right sides are entered.

```
Private Sub btnPercent_Click()
If Not bLeft Then
If sRight <> "" Then
iRight = sRight
Else
iRight = 0
End If
iRight = iRight * (iLeft / 100)
tbResult.Text = iRight
If iRight <> 0 Then
sRight = iRight
Else
sRight = ""

```



```
End If
```

```
End If
```

```
End Sub
```

Lastly, the 1/x, or fraction, Click event, which is very similar to the code above:

```
Private Sub btnFraction_Click()
```

```
If sLeft <> "" Then
```

```
iLeft = sLeft
```

```
Else
```

```
iLeft = 0
```

```
End If
```

```
If sRight <> "" Then
```

```
iRight = sRight
```

```
Else
```

```
iRight = 0
```

```
End If
```

```
If bLeft Then
```

```
iLeft = 1 / iLeft
```

```
tbResult.Text = iLeft
```

```
Else
```

```
iRight = 1 / iRight
```

```
tbResult.Text = iRight
```

```
End If
```

```
If iLeft <> 0 Then
```

```
sLeft = iLeft
```

```
Else
```

```
sLeft = ""
```

```
End If
```

```
If iRight <> 0 Then
```

```
sRight = iRight
```

```
Else
```

```
sRight = ""
```

```
End If
```

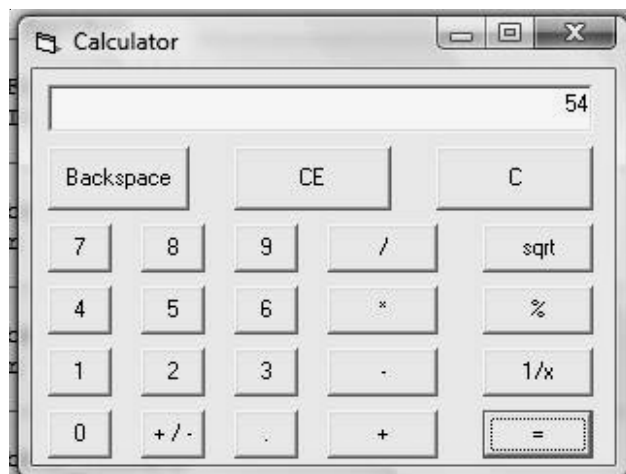
```
End Sub
```

**11:** Add code to handle the C and CE buttons. C clears all input to the calculator, whereas CE only clears the number currently being entered.

```
Private Sub btnC_Click()
 sLeft = ""
 sRight = ""
 sOperator = ""
 tbResult.Text = "0"
 bLeft = True
End Sub

Private Sub btnCE_Click()
 If bLeft Then
 sLeft = ""
 Else
 sRight = ""
 End If
 tbResult.Text = "0"
End Sub
```

**12:**



**Fig. 14.11:** Completed calculator in action (ALLD)



*Caution*

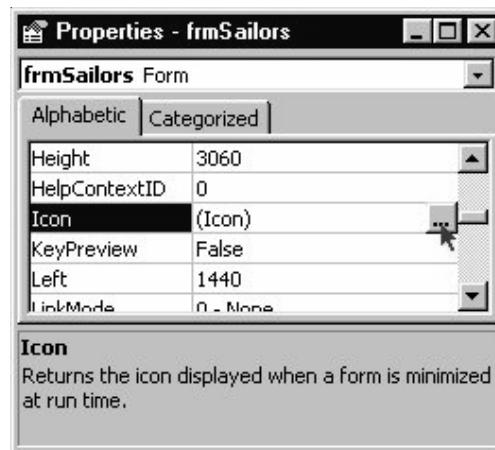
If buttons and textbox are not named the same as the code expects, it will need to change the names to match buttons and textbox, or change buttons and textbox to match this code.



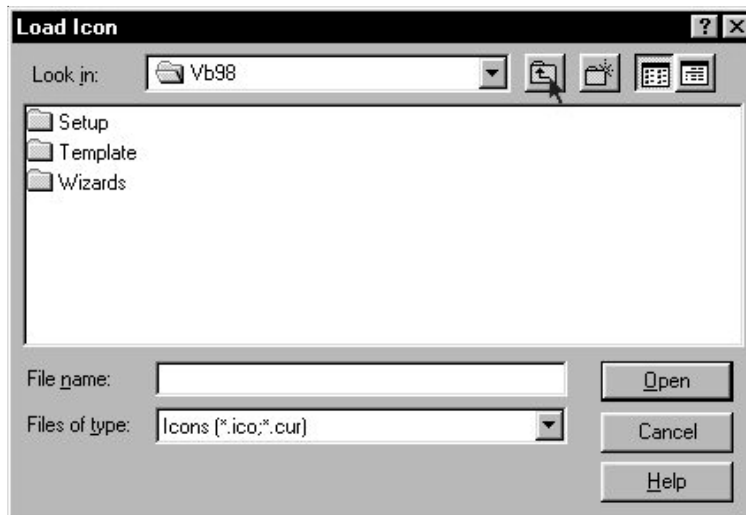
## Case Study

Where do I find icons?

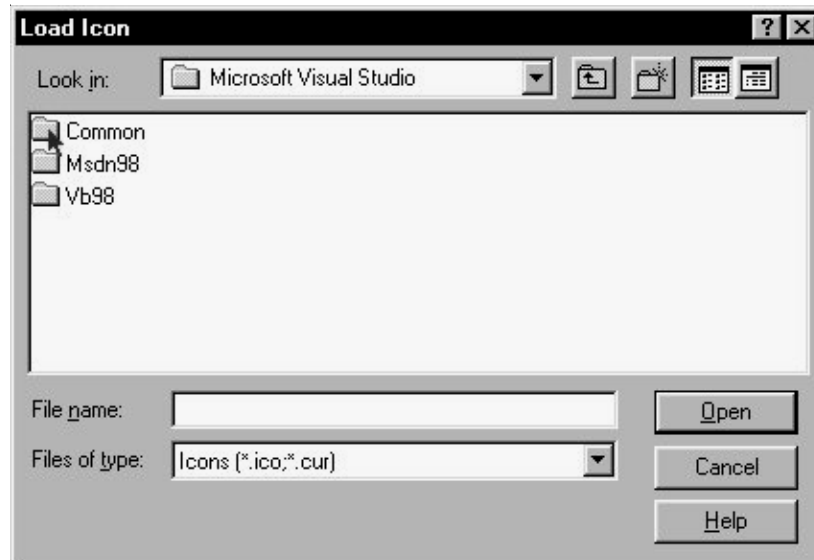
In the Properties box, select the Icon property



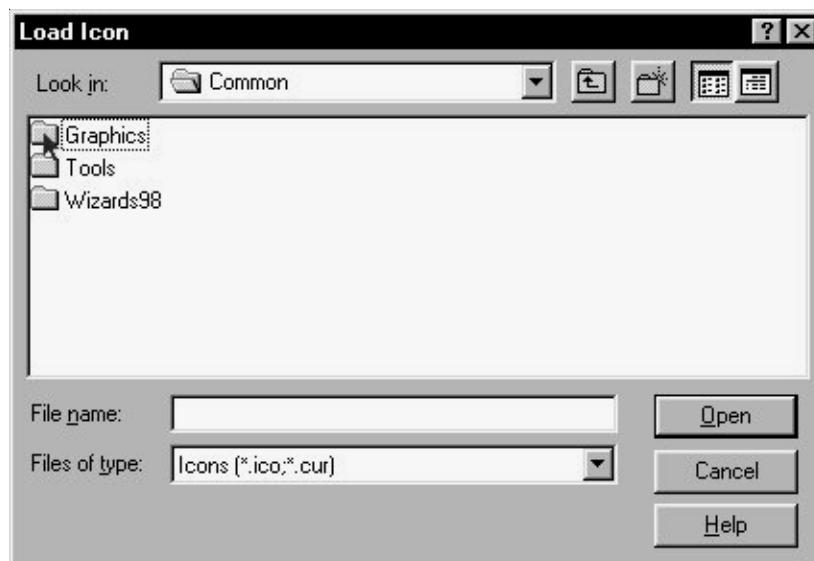
In the Load Icon dialog box, navigate up one folder.



Move into the Common folder.



Move into the Graphics folder.



Move into Icons folder.



In the Icons folder, we will find several folders that names categories. The icons are grouped by category in each folder. Move into each folder and choose the icon we want. Then click Open.

#### Questions

1. Give steps for change property of icons.
2. How to add image in forms background?

### 14.5 Summary

- Building small application, in which we create visual basic projects and visual basic forms.
- Sample project specification covered few specification such as a need for menu function.
- Visual basic controls are objects that determine what the form does, labels, buttons, textboxes are all controls.

### 14.6 Keywords

**A Text box:** Is a common element of graphical user interface (GUI) of computer programs, as well as the corresponding type of widget used when programming GUIs.

**Standard EXE:** It is one that is created using Standard EXE project. It is the most widely used Project type using VB6. Standard EXE application is normally the most widely used among the available Project types in Visual Basic. Stand-alone programs have an .EXE file extension.



Lab Exercise

1. Explain the work of Project listing and properties window (ALLD).
2. Explain all steps in visual basic form creation.

### **14.7 Self Assessment Questions**

1. Write application used to create visual basic programme
  - (a) Block
  - (b) Squire
  - (c) Form
  - (d) None of the above
2. Write visual basic objects which determine what the form does
  - (a) Labels
  - (b) Form
  - (c) boxes
  - (d) none of the above
3. Menu functions need to make able part of form
  - (a) True
  - (b) False
4. Microsoft Visual Basic is a programming language based on the BASIC language
  - (a) True
  - (b) False
5. Project listing and properties window (ALLD) Rename project and form by clicking on "Project1".
  - (a) Right
  - (b) Left
6. When the user hits the.....and the subtotal, the tax payable and the final total will appear.
  - (a) 'Calculate' button
  - (b) Subtotal
  - (c) Credit'
  - (d) None of the above
7. Building a Calculator with Visual Basic calculation consists of how many parts?
  - (a) One
  - (b) Two
  - (c) Five
  - (d) Four

### **14.8 Review Questions**

1. How will you create visual basic project?
2. How will you create visual basic form?
3. Explain Sample project specifications.
4. Explain instructions for creating a calculator with visual basic.
5. How to set the default button for a form?
6. How to set the Cancel button for a form?

**LOVELY PROFESSIONAL UNIVERSITY**

Jalandhar-Delhi G.T. Road (NH-1)  
Phagwara, Punjab (India)-144411  
For Enquiry: +91-1824-521360  
Fax.: +91-1824-506111  
Email: [odl@lpu.co.in](mailto:odl@lpu.co.in)

978-93-87034-96-9



9 789387 034969

7. How do I use charts in Visual Basic?
8. How do I conditionally format a TextBox?

### Answers for Self Assessment Questions

1. (a)
2. (a)
3. (a)
4. (a)
5. (a)
6. (a)
7. (a)

### 14.9 Further Readings



*Books*

G. Cornell, "Visual Basic 6", Tata McGraw-Hill, 1998.

Null Dale, Michael Mc Millan, "Visual Basic .Net".



*Online link*

<http://microsoft.com/mspress/findabook/list/title.aspx>.