# VISUAL PROGRAMMING USING VB

## SYLLABUS

### UNIT I

Introduction to Window's 2000 – Operating System Features – Basic Operations – Fundamental of Visual Basic : Anatomy of Visual Basic Program – The Code Window – Statements in Visual Basic – Assignment and property setting – Variables – Strings – Number constants, repeating operations, making decisions.

### UNIT II

Working with object at run time – Projects with multiple forms – Displaying information. The Printer object – Advanced programming techniques – Arrays – Pointers – Built in functions – User defined functions & procedures.

### UNIT III

Objects – Manipulation of objects in Visual Basic – Collections – Creating an object in Visual Basic – Building classes – Files – Sequential files – Random access files – Binary files – Sharing files.

### UNIT IV

Communicating with other windows application: Clip board – Activity windows application – Dynamic data exchange & OLE 2.

Data base Featues: Modern databases – Data manager – Using the data control – Programming with data control – Monitoring changes to the database – SQL basics – Data base objects ADO.OLE.DB.

### UNIT V

Printing – Reports – Writing – Error handlers – Debugging techniques – DHTML – Internet/Intranet Applications using Visual Basic – Active X documents – Winsock Control.

# UNIT I

# LESSON

# 1

# INTRODUCTION TO WINDOWS 2000

CONTENTS



## 1.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the operating system features
- Discuss basic operations in operating system

## 1.1 INTRODUCTION

In the earlier days of Windows-based programming, the programmers were a highly confused and stressed group. For Windows-based programming, the programmers had to master the Microsoft Windows SDK (Software Development Kit). Of course, they had to write all the code. However, besides the routine code, the programmers also had to code for the entire GUI (Graphical User Interface). This meant that programmers had to draw a textbox, a line, a button, etc. all by coding. The application was then compiled and run. When the application ran, the developer could see the GUI. Once, any changes were needed in the GUI, the developer had to stop the program, come back to the code window and make the desired changes in the code. This way, the developers were constantly shuttling between:

- The editor,
- The compiler, and
- Any other tool for GUIs (if available).

Obviously, constantly shifting from one environment to another was a bit of a problem. There was no convenient option of a drag-and-drop for drawing the GUI. Microsoft sensed this problem and launched the language Visual Basic (VB). With VB, developers had a single window for all the needs:

- Designing the GUI,
- Editing the code, and
- Compiling the code.

This way, the concept of an IDE (Integrated Development Environment) was born. We can think of the IDE as a 'one-stop-shop' for all the programming needs of programmers. With the IDE in place, the programmers did not have to keep shifting between different applications while programming - a single window was all they needed. This window was Visual Basic, version 1.0.

VB is also commonly referred to as a RAD (Rapid Application Development) tool. This is for the reason that with VB, we can develop an application really fast. In fact, the prototype can be made in just a few hours. However, RAD should not mean a compromise in proper application design. This was very clear to the Microsoft team that developed VB. Thus, VB offers an increasingly complex set of attributes based on today's client/server architecture, including Internet/Intranet based application development. It has grown a lot in maturity since its humble beginning as a simple programming language.

## 1.2 WINDOWS 2000

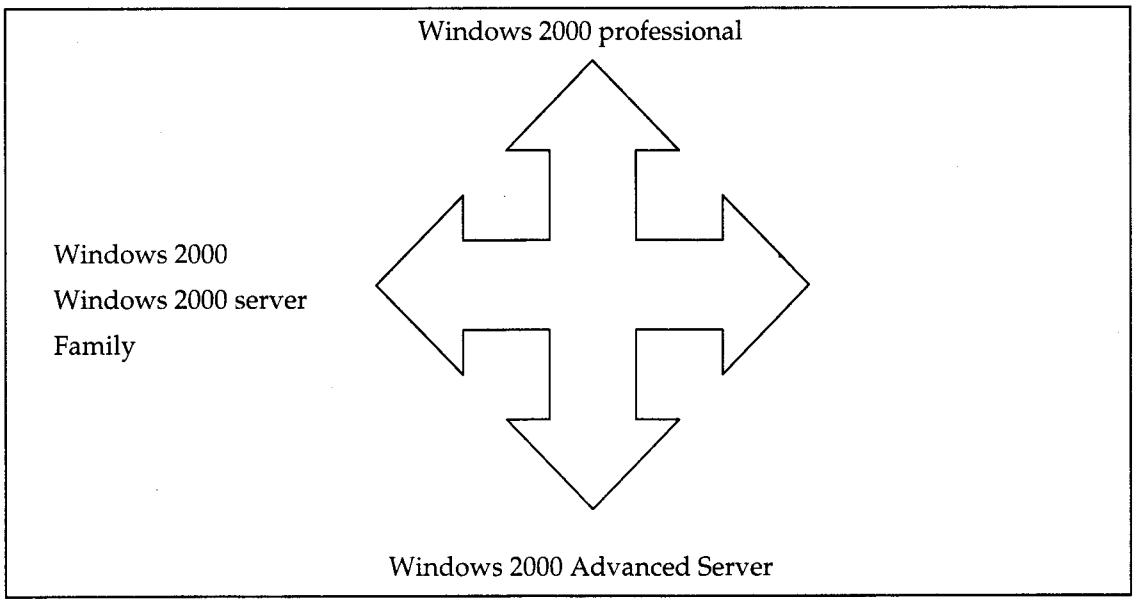
A family of operating systems for personal computers, Windows dominates the personal computer world, running, by some estimates, on 90% of all personal computers. Like the Macintosh operating environment, Windows provides a Graphical User Interface (GUI), virtual memory management, multitasking, and support for many peripheral devices. In addition to Windows 3.x and Windows 95, which run on Intel-based machines, Microsoft also sells Windows NT, a more advanced operating system that runs on a variety of hardware platforms.

Windows 2000 is the name given to the next version of Microsoft's line of operating systems; formerly know as Windows NT 5.0. In the future, the core code of NT will serve as the basis for all of Microsoft's PC operating systems—from consumer PCs to the highest performance servers. In fact, the next major release for consumers to follow Windows 98 will be based on the Windows NT code base. This name will encompass all future business and consumer releases of Windows.

Microsoft Windows 2000 (Professional, Server, and Advanced Server) is a server and workstation operating system made by Microsoft that runs on Intel/Cyrix/AMD Pentium.

Windows 2000 is Microsoft's third attempt to provide a reliable desktop operating system. The first attempts were Windows 98 and Windows NT. Windows 2000 was originally planned to combine the ease of use of Windows 98 with the supposed "reliability" of Windows NT, but Microsoft still was unable to accomplish that modest goal and announced plans for a continued two track system (Windows 2000 for "professional" use and Windows Millennium Edition (ME) for desktop use).

## 1.2.1 Windows 2000 Family




## 1.2.2 Windows 2000 Professional

Windows 2000 Professional will be Microsoft's mainstream desktop operating system for businesses of all sizes, replacing Windows NT Workstation 4.0, which many people are using today as the standard business desktop. Windows 2000 Professional will deliver the easiest Windows yet, the highest level of security, state-of-the-art features for mobile users, industrial-strength reliability, and better performance while lowering the total cost of ownership through improved manageability.

Windows 2000 Professional is the Windows operating system for business desktop and laptop systems. It is used to run software applications, connect to Internet and intranet sites, and access files, printers, and network resources.

Windows 2000 Professional is designed for business desktops and corporate mobile users that require access to their data and personalized settings regardless of physical location. Unlike Windows 98, which caters to the home/consumer and gaming markets, Windows 2000 Professional - which replaces Windows NT Workstation - is designed solely for businesses, which its feature-set bears out. Still, so-called power users and software developers will probably flock to Windows 2000 Professional as they did with Windows NT Workstation. And they should: This is the best desktop operating system that Microsoft has ever created.

Built on Windows NT® technology and the easy-to-use, familiar Windows® 98 user interface, Windows 2000 Professional gives business users increased flexibility. The integrated Web capabilities let you connect to the Internet from anywhere, at anytime—giving your company access to host of flexible, cost-effective communications options. In addition, broad peripheral and mobile computer support make Windows 2000 Professional an ideal operating system for a workforce that increasingly relies on notebook computers. Further, your support and administrative staff will particularly appreciate the reliability and manageability enhancements that make desktop management simpler and more efficient. Windows 2000 Professional lets you.

The Windows 2000 Professional operating system naturally integrates using the Internet with the daily tasks. From working with files and Web pages to sharing information with fellow employees or business partners, the flow of work between desktop, network, and the Internet is more seamless: you don't have to remember several ways of doing things.

Small businesses can benefit from windows 2000 professional from the ability to connect several people on the Internet through one computer.

Large businesses can take advantage of improved communications management made possible using Windows 2000 Professional on their client computers.

All sizes of companies can use the standards-based technology in Windows 2000 to create custom applications that strengthen the connections between employees, customers, and partners.

## 1.3 OPERATING SYSTEM FEATURES

*Tight Browser Integration*

With Microsoft Internet Explorer 5.01 built in, Windows 2000 Professional offers you the easy way to connect and work on the Internet, whether at the office, at home, or even offline. Internet Explorer 5.01 is designed to save time on the two most common tasks—searching and keeping track of information. It helps in finding and organizing information on your hard drive, intranet, or Internet.

With the Search bar, one can search faster and get exactly where he/she wants to go. One can choose whether to search Web pages, people's addresses, or businesses and which search engine to use. With the integrated History bar, you can quickly find your way back to information you browsed in the past—whether it was on a Web site, intranet site, or in a file folder. All information is listed in the history in the same way, so you don't have to hop around between file systems and favorites lists, or run new searches to find material you've already located.

*Multiple Machines Share a Single Internet Connection*

For businesses that want to provide access to a few employees with minimal hardware expense, Windows 2000 professional allows up to 10 computers to simultaneously share a single connection—either dial-up or broadband—to the Internet. This capability, provided by the Internet Connection Sharing feature, is ideal for small businesses.

*Telecommuters and Mobile Users Connect using the Internet*

It's only natural to want to be able to use your corporate network and the Internet just as easily when you're away from the office as you can when you're at your desk. Regardless of where the person is, Windows 2000 professional can help you connect quickly and easily with security in place. One can connect directly to the company network with a dial-up connection or ISDN line whether he/she is in the field or using the home computer. Connecting to the network through the Internet is another option using the Virtual Private Networking (VPN). This creates a security-enhanced communications link between the computers and, even if one is using the Internet rather than a private phone line or network cable.

*Voice, Video and Data Networks Integrated with the Internet*

The Internet and network convergence can significantly change the way business provides service to the customer. The customers can interact with business using phone, e-mail, fax, chat, Instant

Messaging, and Voice-over IP. One can use this mix of communication technologies to provide real-time customer service on the Web to improve the sales efforts, and particularly e-commerce activities.

*Strong Development Platform*

The telephony technologies are a prime example of the features in Windows 2000 Professional that developers can use to create custom applications for the business. Using Windows 2000 Professional as a client gives application developers several advantages. With support for Dynamic HTML (DHTML) and DHTML Behaviors in Internet Explorer 5, developers can build rich applications with attributes that can be easily replicated and tested on multiple sites, faster than ever before. Plus, with the browser support for Extensible Markup Language (XML), developers can invent new ways to create, exchange, and display information.

*File System Support in Windows 2000*

Windows 2000 systems can support the following file systems:

- FAT, FAT32
- NTFS - New Technology File System
- CDFS - Compact Disk File System
- UDF - Universal Disk Format for DVDs.
- EFS - Encrypting File System runs as a service and is used to encrypt and decrypt files on an NTFS file system for security purposes. The EFS is not a file system like NTFS since it does not create partitions and control the placement of file data, it only is used to control the encryption of data. See the Section called "Security" in this document for more information on NFS.

*FAT Filesystem Characteristics*

Used with DOS, it can only support partitions up to 4 G. No spaces are allowed in the file name.

*FAT32 or VFAT Filesystem Characteristics*

VFAT - Virtual File Allocation Table introduced by Windows 95 that allows long file names. VFAT is not natively supported by Windows 2000.

- FAT32 filesystems support partitions up to 32GB.
- Filenames up to 255 characters long.
- Filenames begin with a letter and exclude " / \ [ ] : ; | = , ^ * ?
- The last part is the extension but spaces can be used
- It supports file attributes used by DOS such as read-only, archive, system, and hidden.
- Won't support running POSIX applications.

FAT partitions provide no local security, only share level security across a network.

*NTFS Filesystem Characteristics*

Windows 2000 NTFS file systems are newer than Windows NT NTFS file systems. In order for Windows NT and Windows 2000 to use the Windows 2000 file system together, the Windows NT system must have service pack 4 or later installed.

- Filenames up to 255 characters long
- Filenames preserve case but are not case sensitive.
- Filenames exclude " / \ < > : | * ?
- Supports built in file compression as a file attribute. Compression is applied to files in a folder if that folder has its compression attribute set. Also optionally sub folders and their contents may be compressed. Compression is not supported if the cluster size is above 4K in size. Moved files retain their compression attribute, but if they are copied they will assume the compression attribute of the target folder.
- Provides automatic transaction tracking of disk activity for correcting corrupt or failed operations.
- Supports auditing.
- Provides sector sparing.
- There is a recycle bin for each user.
- Windows 16 bit and DOS environments can't use this filesystem.
- A master file table is used to save individual file, boot sector, disk structure, and file recovery information.
- Automatically makes 11 character DOS file names. When the first 8 characters of long filenames match, the first four DOS file names use the first for characters of the long name, the ~ and 1, then 2, etc. After the fourth duplicate name, the first two characters are used, then the next four characters are hashed, then the ~ character then a number. The first two duplicate file names may be: DOCU~1.DOC and DOCU~2.DOC. The long extension is used as part of the extension for the 8.3 filename alias. Directory entries used by long filenames include 1 for the 8.3 alias and 1 for each 13 characters in the long filename.
- Provides file logging ability and file recovery.
- Supports POSIX.
- Maximum file or partition size of 16 exabytes.
- Supports file sharing with MacIntosh clients.
- The disk is in 8M bands with a 2K file allocation map between each band. The 2K map is a map for the associated 8M band. This structure is called the BTREE and is used to reduce fragmentation.
- Supports file encryption with the Encrypting File System (EFS) on Windows 2000.
- Allows volumes on remote computers or local computers to be mounted as though they are part of the same partition they are mounted on. This feature is available on Windows 2000.

- Disk quotas (tracking of disk space) on a user by user basis are tracked.
- Removable media formatted in NTFS can be changed and accessed without rebooting the system in Windows 2000 (not NT).

If installing DOS with NT, install DOS first so DOS will not corrupt the NT boot sector and stop the NT boot manager from running. Floppies are formatted as FAT, not NTFS.

*CDFS*

The file system that supports Compact Disks (CDs) is the Compact Disk File System (CDFS).

*UDF*

The file system that supports DVDs is the Universal Disk Format (UDF).

The FAT file system does not support file compression on Windows 2000 systems. The file compression utilities with Windows 95 and Windows 98 are not supported by Windows 2000. FAT file systems may be converted to NTFS file systems using the command line convert utility. Once converted, they may not be changed back to FAT.

Windows 2000 contains an NTFS file defragmentation utility, which Windows NT does not contain.

*Support for Security*

Each object has an Access Control List (ACL) which defines users and group permissions for the object. Each entry (ACE - Access Control Entry) in an ACL defines the permissions a specific user or group has for the object. Access token attributes are added to the object's ACL. The user's security identifier (SID) is compared to the contents of the ACL to determine if the user has the correct privileges to access the object.

The NTFS file system supports Access Control Lists for objects.

*All the Advantages of the Windows 2000 Server Family*

Windows 2000 Datacenter Server shares the following features with the Windows 2000 Server Family:

- Windows 2000 Active Directory Service centrally manages Windows-based clients and servers through a single consistent management interface, reducing redundancy and maintenance costs.
- Microsoft IntelliMirror™ management technologies install and maintain software, apply correct computer and user settings, and ensure that users' data is always available.
- Group Policy allows central management of groups, computers, applications, and network resources, instead of managing entities on a one-by-one basis. Group Policy's integration with Active Directory delivers more granular and flexible control.
- Microsoft Internet Information Services (IIS) 5.0 enables users to easily host and manage websites.
- Cluster Service supports 4-node fail over support for critical applications.
- Network Load Balancing (NLB) redistributes workload among remaining servers in less than 10 seconds in the event of a hardware or software failure on one of the servers.
- Public Key Infrastructure (PKI) and Certificate Services: The Certificate Server is a critical part of PKI that allows customers to issue their own x.509 certificates to their users for PKI functionality such as certificate-based authentication, IPSec, or secure e-mail.

- Remote Access Service connects remote users to the corporate network through dial-up, leased lines, and Internet links.
- Virtual Private Network (VPN) support, with a full-featured gateway that encrypts communications to securely connect remote users and satellite offices over the Internet.
- Terminal Services makes it possible for networks to run Windows-based applications on the server, and allows clients to access them from a remote PC, Windows-based terminal, or non-Windows device over LANs, WANs, or low-bandwidth connections.
- File and print sharing services provide a unified file and print infrastructure for securely sharing, storing, and publishing information.

Microsoft will continue to innovate and improve its datacenter platform with future operating system releases.

## 1.4 BASIC OPERATIONS

Windows 2000 Professional versus Windows 3.1, Windows 2000 Professional versus Windows 95/98, Windows 2000 Professional versus Windows NT Workstation 4.

*Windows 95 Operating System*

- *Background:* Introduced in 10/1995
- *Support:* Not available. Not supported after 12/2001
- *Requirements:* 486-25Mhz, 16-32MB of memory, 50MB-600MB of disk space
- *Features:* FAT32 file system was introduced with OSR2 (W95b). Because it was still based on DOS, it allowed more backwards compatibility than NT at the time it was introduced.
- *Security:* Password protection is easy to override
- *Upgrade:* Cannot be upgraded to any version of XP

*Windows 98 Operating System*

- *Background:* Released before 7/1999 Basically an updated 95 version.
- *Support:* Not available since 6/2002. Not supported after 7/2003
- *Requirements:* 486-66MHz, 32-256MB of memory. Provides support for DVD and USB
- *Features:* Based on DOS. Uses the FAT32 file system. It allowed more backwards compatibility than NT versions. It can automatically check and download system updates from the Internet
- *Security:* Password protection is easy to override
- *Upgrade:* Can be upgraded to XP

*Windows NT Server*

- *Background:* Introduced in 7/1999.Combination of NT 3.5 technology with an interface based on 95.
- *Support:* Not available since 7/2002. Support till 1/2005.

- *Requirements:* Pentium-100Mhz, 32-64MB of memory, 110MB-1GB of disk space. Has poor or no support for laptop power management features. 32-256MB of memory. Provides support for DVD and USB
- *Features:* Using 32bit architecture. It is slightly more complicated to administer than 95, but gives greater options for networking and machine security. Because it is not based on DOS it does not suffer from backwards compatibility problems but can cause trouble for some DOS-based legacy applications. Uses the FAT32 or NT File System, which is more robust than standard DOS FAT file system. Allows file and directory level Access Control Lists (permissions) with inheritance and a smaller sector size using less disk space.
- *Security:* Entire system is password protected with multiple user and group policy security configuration
- *Upgrade:* Can be upgraded to XP Pro, but NOT XP Home.

*Windows 2000 Professional*

- *Background:* Introduced in 4/2000
- *Support:* Available until 4/2003.Not supported after 4/2004.
- *Requirements:* Pentium133Mhz64MB of memory650MB-2GB of disk space.
- *Features:* U Contains all of the features of NT 4.0Supports up to two processors. Does not automatically check and download updates. SysPrep utility allows cloning of computer configurations, systems and applications and can also be installed remotely.
- *Security:* Entire system is password protected with multiple user and group policy security configuration. Built in VPN, using IPSec and required at both ends of the connection, is available to encrypt data sent over the Internet.
- *Upgrade:* Can be upgraded to XP Pro, but NOT XP Home.

*WIN ME*

- *Background:* Introduced in 9/2000.Looks like 2000.Based on 9xIntended for home users.
- *Support:* Available until 9/2003.Not supported after 9/2004.
- *Requirements:* Pentium 150Mhz32-128MB of memory.320MB-2GB of disk space.
- *Features:* Can automatically check and download system updates from the Internet has poor network and hardware support and stability issues. Most hardware manufactures ignore ME when writing new drivers.
- *Security:* Entire system is password protected with multiple user and group policy security configuration. Built in VPN, using IPSec and required at both ends of the connection, is available to encrypt data sent over the Internet.
- *Upgrade:* Can be upgraded to XP Pro or Home

*WIN XP*

- *Background:* Introduced in 10/2001
- *Support:* Available after 10/2004Probably not supported after 10/2005.

- *Requirements:* Pentium 300MHz 128MB memory 1.5GB disk space.
- *Features:* Microsoft's first OS to combine the 9x code with the NT code, removing the MSDOS layer from under Windows 9x.New visual design with visual cues and redesigned Start menu. Dual View Desktop allows display on 2 monitors. Supports up to 2 processors and up to 4GB of RAM. Latest standards for DVD, Infrared Data, USB and high speed bus IEEE1394. Integrated support built-in for burning CD-R's and CDRW's. Improved power management for notebooks. Remote Desktop allows you to run your computer from another computer. Remote clients must have 95 or later. Remote Assistance to share control of their computer with someone on the network or Internet. Requires XP at each end. Offline files allow a user to specify which network based files are needed to run remotely. Synchronization manager for comparison of offline files those on the network. Network Location Awareness determines changed network locations. Ability to restore a system to a previous state without losing data. Although, this usually causes more problems than it prevents. Don't recommend using it. Integrated service helps install, configure, track, upgrade and remove software programs correctly. Multiple versions of DLL's to allow different versions of applications to run side-by-side. Compatibility mode for older applications to run in NT 4.0, 95, 98 or ME mode. Prevents the core system from being overwritten by application install. Eliminates almost 99% of rebooting requirements when installing software applications. Maintains a copy of previously installed drivers, which can be reinstalled if problems occur. Can automatically check and download system updates from the Internet. SysPrep allows cloning of computer configurations, systems and applications and can be installed remotely.
- *Security:* Entire system is password protected with multiple user and group policy security configuration. Built in VPN, using IPSec and required at both ends of the connection, is available to encrypt data sent over the Internet. File encryption is available to protect privacy.
- *Upgrade:* Can be upgraded to XP Pro or Home

### 1.4.1 Windows NT Workstation

*User Interface*

| Feature | Description |
|---|---|
| Windows 95 User Interface | Windows NT Workstation 4.0 has the same interface as Windows 95. This means that you can use the same interface for all your Windows-based 32-bit desktops and servers. The Windows 95 interface includes:<br>Start Button, Taskbar, shortcuts<br>My Computer, Network Neighborhood, and the Recycle Bin |
| Windows NT Explorer | Windows NT Explorer is a tool for browsing and managing files, drives, and network connections. Windows NT Explorer displays your computer's contents as a hierarchy, or "tree," enabling you to see the contents of each drive and folder on your computer, as well as any network drives your computer is connected to. |

Windows NT Explorer replaces the File Manager, which was used in previous Microsoft Windows operating systems.

Installation

The installation process simplifies the setup procedure when upgrading to Windows NT Workstation. Features include an interface, hardware detection, installation wizards, and a series of tools for corporate customers to deploy Windows NT Workstation on multiple systems.

Accessories

The Windows NT Workstation 4.0 operating system includes a number of additional applications and utilities such as:

(a) HyperTerminal, a 32-bit communications application that provides asynchronous connectivity to host computers such as online services. HyperTerminal is pre-configured to allow access to AT&T Mail, CompuServe, MCI Mail, and other systems.

(b) WordPad, a 32-bit editor that allows users to create simple documents and memos.

(c) Paint, a 32-bit graphics application that allows users to read PCX and BMP file formats.

(d) Quick Viewers, enabling users to view files in the most popular file formats without opening the application that was used to create the file.

*Access to the Internet and Corporate Intranets*

Microsoft Internet Explorer

Microsoft Internet Explorer is Microsoft's easy-to-use Internet browser.

Peer Web Services

Microsoft Peer Web Services (PWS) enables publication of personal Web pages. PWS allows users to share information on their corporate intranets. PWS is ideal for development, testing, and staging of Web applications, as well as peer-to-peer publishing. As with Windows NT Server's built-in Web server, Microsoft Internet Information Server (IIS), PWS supports all ISAPI extensions and filters. PWS has been optimized for interactive workstation use, and does not have the system requirements (memory requirements, server processes, and footprint) of a full Web server such as IIS.

Client Support for PPTP

The Point-to-Point Tunneling Protocol (PPTP) provides a way to use public data networks, such as the Internet, to create virtual private networks connecting client PCs with servers. PPTP offers protocol encapsulation to support multiple protocols via TCP/IP connections, and data encryption for privacy—making it safer to send information over non-secure networks. This technology

extends the Dial-up Networking capability by enabling remote access and securely extending private networks across the Internet without needing to change the client software.

WINS and DNS Integration

Windows NTWorkstation4.0takesadvantageof the integration between two Windows NT Server services—Windows Internet Name Service (WINS) and Domain Name System (DNS)—to provide a form of dynamic DNS. With WINS and DNS integration, users can enter DNS fully qualified domain names, making it easier to connect to network resources. For example, using the Windows NT Explorer, a user could gain access to a share via a DNS name such as \\srv1.myco.com\public.

### *Network Integration*

Network Control Panel

The Windows NT Workstation 4.0 Network Control Panel provides a single access point where all network settings—such as, identification, services, protocols, adapters, and bindings—can be installed and configured.

### *Management Features*

System Policies and User Profiles

System policies and user profiles allow system administrators to manage and maintain their users' desktops in a consistent manner. System policies are used to standardize desktop configurations, to enforce behavior, and to control users' work environments. User profiles contain all user-definable settings for the work environment of a computer running Windows NT Workstation 4.0. Profiles can be stored on a Windows NT Server, so users always receive the same desktop when logging on to any Windows NT-based computer on the network.

Setup Manager

Setup Manager is a utility that assists system administrators in creating installation scripts, thereby reducing the time and effort of deploying Windows NT Workstation 4.0. The new Setup Manager provides a graphical interface for creating hands-free installation scripts that allow system administrators to automate installation for end users. These hands-free scripts eliminate the need for users to answer questions during the installation process, thus avoiding mistakes that can occur during system software upgrades.

System Difference Utility

Windows NT Workstation 4.0 includes the System Difference Utility (sysdiff) that provides a way to pre-install additional applications simultaneously with the operating system. The sysdiff utility allows system administrators to create packages that can be applied to a system during installation. These packages can also be applied during Windows NT Workstation 4.0 setup.

| | |
|---|---|
| Windows NT Diagnostics Program | Windows NT Workstation 4.0 includes a Windows NT Diagnostics program that simplifies troubleshooting. It contains information such as build number, device driver information, network usage *data, and information about system resources like IRQ, DMA, and* I/O address. Diagnostics information is viewed in a graphical tool that you can also run remotely on Windows NT. |
| Printer Management | Printer management allows printers to be managed remotely |

***Networking and Windows 2000 Professional***

| | |
|---|---|
| Dial-Up Networking | Windows NT Workstation 4.0 extends the functionality offered by Dial-Up Networking and provides the ability to automatically dial a connection when required. With automatic dialing, dial-up networking is smoothly integrated into the new Windows NT interface. Whether users are connecting to the Internet, running client/server applications, accessing remote databases, or accessing shared files, mobile network access is as easy as network access in the office. Establishing a remote connection works the same as establishing a local connection—simply |
| Multimedia and Graphics Multimedia APIs | Windows NT Workstation 4.0 supports the multimedia APIs first introduced in Windows 95: DirectDraw®, DirectInput®, DirectPlay®, and DirectSound®. Supporting these APIs allows developers to simultaneously create games and other applications for Windows 95 and Windows NT Workstation 4.0 platforms. |
| Imaging for Windows NT | The Microsoft imaging software for Microsoft Windows 95 is available on Windows NT. This imaging software provides imaging services that enable users to access and control information directly at their desktops. |
| Compact Disc File System | Windows NT Workstation 4.0 now supports the following |
| Enhancements | Compact Disc File System (CDFS) enhancements: Auto-Run and CD-XA formats. Auto-Run allows the operating system to recognize that a compact disc has been inserted into the drive and to start the application immediately. CD-XA is an extended format for video compact discs that contain MPEG movies. |
| Driver Support | Windows NT Workstation 4.0 includes numerous video drivers that improve screen quality and are especially helpful when using multimedia features. Some of the newly supported drivers are: WD ThinkPad, Matrox Millennium, Trident, Number 9 Imagine, C&T, and Cirrus. |

## *Application Programming Interfaces and Additional Features*

| | |
|---|---|
| Telephony APIs | Telephone API (TAPI) integrates advanced telephone capabilities with the power of PCs. TAPI provides a level of abstraction software developers so their applications don't need to be bound to specific telephone hardware. Through the TAPI interface, communications applications can ask for access to the modem or telephone device. Unimodem (Universal Modem Driver) provides TAPI services for data/fax modems and voice so that users and application developers don't have to learn or maintain difficult modem "AT" commands to dial, answer, and configure modems. Some of the Windows NT 4.0 applets that use the TAPI/Unimodem support are Dial-up Networking, HyperTerminal, and Phone Dialer. Cryptography APIs Windows NT Workstation 4.0 includes a set of encryption APIs that allow developers to create applications that work securely over non-secure networks, such as the Internet. |
| Distributed COM | The Component Object Model (COM) allows software developers to create component applications. Now, Distributed COM (DCOM) in Windows NT Workstation 4.0 and Windows NT Server 4.0 provides the infrastructure that allows DCOM applications (the technology formally known as Network OLE) to communicate across networks without needing to re-develop applications. An example of a DCOM application would be a stock quote server object running on Windows NT Server that distributes quotes to multiple clients running Windows NT Workstation. DCOM provides the infrastructure for connecting and providing uniform communication between client/server objects. DCOM uses the same tools and technologies as COM, preserving investments in training and software. |
| 486 Emulator | Allows 386-enhanced 16-bit applications to run on RISC machines |

**Check Your Progress**

Fill in the blanks:

1. Windows 2000 ..... offers you the easy way to connect and work on the Internet, whether at the office, at home, or even offline.
2. Connecting to the network through the Internet is another option using the .....
3. Encrypting File System runs as a service and is used to encrypt and decrypt files on an ..... file system for security purposes.
4. Windows NT Explorer is a tool for ..... and managing files, drives, and network connections.

## 1.5 LET US SUM UP

Windows is a family of operating systems for personal computers. Microsoft Windows 2000 (Professional, Server and Advanced Server) is a server and workstation operating system that runs on Intel/Cryix/AMD Pentium. Windows 2000 Professional is the windows operating system for business desktop and laptop systems. It is used to run software applications, connect to the Internet and intranet sites and access files, printer and network resources. Windows File Protection, a of the features of Windows 2000 Professional, can detect if a system file has been changed or deleted, retrieve the correct version of the file from the cache and restore it to the system file folder. Windows 2000 Server is the multipurpose network operating system for business of all sizes. It offers file and printer sharing reliably and securely, the flexibility of choosing from thousands of business applications compatible to run on the server, build web applications and to connect to the Internet. With Windows 2000 Server, Microsoft introduces a new administrative method for networks which is the Active Directory. Windows 2000 Advanced Server is the server operating system for line of business applications and e-commerce. It has additional scalability and reliability features such a clustering, designed to keep the business critical applications up and running in the most demanding scenarios.

## 1.6 KEYWORDS

*Windows:* Windows is a family of operating systems for personal computers.

*Microsoft Windows 2000:* (Professional, Server and Advanced Server) is a server and workstation operating system that runs on Intel/Cryix/AMD Pentium.

*Windows 2000 Professional:* It is the windows operating system for business desktop and laptop systems.

*Windows 2000 Server:* It is the multipurpose network operating system for business of all sizes.

*Windows 2000 Advanced Server:* It is the server operating system for line of business applications and e-commerce.

*Windows File Protection:* It can detect if a system file has been changed or deleted, retrieve the correct version of the file from the cache and restore it to the system file folder.

## 1.7 QUESTIONS FOR DISCUSSION

Select the most appropriate answer.

1. You are choosing a Windows 2000 operating system to use on a new computer at your company. This new computer will be used exclusively as an employee's desktop computer. Which operating system should you use?

   (a) Windows 2000 Professional

   (b) Windows 2000 Server

   (c) Windows 2000 Advanced Server

   (d) Windows 2000 Datacenter Server

2. You are choosing a Windows 2000 operating system to use on a new computer at your company. This new computer will be used exclusively as a network file server. Which operating system should you choose?

   (a) Windows 2000 Professional

   (b) Windows 2000 Server

   (c) Windows 2000 Advanced Server

   (d) Windows 2000 Datacenter Server

3. You are choosing a Windows 2000 operating system to use on a new computer at your company. This new computer will be heavily used SQL Server in your enterprise network environment. Which operating system should you choose?

   (a) Windows 2000 Professional

   (b) Windows 2000 Server

   (c) Windows 2000 Advanced Server

   (d) Windows 2000 Datacenter Server

4. Which hardware platform (or platforms) is supported by Windows 2000? (Choose all that apply).

   (a) The Intel Pentium/166 MHz (and higher) platform

   (b) The Compac Alpha platform

   (c) The PowerPC platform

   (d) The MIPS R 4000 platform

   (e) All hardware platforms

**Check Your Progress: Model Answers**

1. Professional
2. Virtual Private Networking
3. NTFS
4. browsing

## 1.8 SUGGESTED READINGS

Jerry Honeycutt, *Introducing Microsoft Windows 2000 Professional*, Microsoft Pr (May 1999).

Sybex, Sybex Inc., *Windows 2000 Complete, Sybex Books*; 1st edition (June 15, 2000).

Microsoft Corporation, *Microsoft Windows 2000 Professional*, Microsoft Press, Rom edition (March 15, 2000).

Microsoft Corporation, *Microsoft Windows 2000 Professional*, Microsoft Press, Package edition (July 2001).

LESSON

# 2

# FUNDAMENTAL OF VISUAL BASICS

CONTENTS



## 2.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the anatomy of visual basics
- Discuss the code window

- Describe the significance of statements in visual basics
- Identify and explain the assignment and property setting
- Discuss the variables and string
- Explain the concept number constants and repeating operations
- Discuss the control statements/making decisions

## 2.1 INTRODUCTION

The first version of Visual Basic, VB 1.0, did not offer anything extra-special as a programming language. In fact, many magazines of that time termed it as an overly simple programming language. However, these journals also mentioned that the IDE provided by VB is a very big achievement. According to these magazines, the IDE itself can be a very major reason to use VB. After the launch of VB 1.0, for the first time, drag-and-drop functionality became available to Windows programmers. In a single stroke, Windows programming became a lot simpler. VB 2.0 also did not offer much new in programming.

Microsoft launched MS-Access after launch of VB 2.0. Thus, starting with VB 3.0, MS-Access was clubbed together with VB as a back-end tool. A special data access technology, DAO (Data Access Objects) was launched to help work on data in and out of MS-Access.

DAO interacted with the JET engine of Access and gave a very good performance where VB was the front-end and MS-Access the back-end. With VB 3.0, DDE (Dynamic Data Exchange) was launched. Through DDE, it became very easy for data to be shared across different Windows applications.

VB 4.0 was another leap forward. It acted as a bridge between 16-bit and 32-bit computing. Upto Version 3.0, all the development in VB was on 16-bit platforms. With Version 4.0, the user had to choose between 16-or-32 bit development at the time of installation of VB. This version also saw the launch of OLE (Object Linking and Embedding). OLE was more powerful and reliable than DDE for data-sharing. VB 4.0 also offered a new technology for accessing databases remotely, the RDO (Remote Data Objects). Version 5.0 went to even greater heights with the introduction of Active-X and COM. Active-X was a refinement of OLE version 2.0. With the introduction of Active-X, VB began to be taken more seriously by programmers the world over.

VB version 6.0 offered a host of new features. Some of these are:

- *ADO (Active-X Data Objects):* ADO is a data access technology that can link up with any kind of a data source: Oracle, Access, SQL Server, and even a text file.
- *Data Reports:* A tool for creating reports. Data Reports is also based on the ADO model. It is available in VB 6.0 in addition to Crystal Reports.
- *DED (Data Environment Designer):* A simple interface for interacting with a back-end. With DED, we can generate a data-bound VB Form with minimum time and effort. With DED, it is very easy to configure and view hierarchical recordsets, all without a single line of code.
- *DHTML (Dynamic HTML) and IIS (Internet Information Server) applications:* With these, we can easily build internet-based applications using the power of VB 6.0.

- *Package and Deployment Wizard:* It helps in organizing and collecting the files required for an application into a single container. Using this utility, we can make an installable CD for our project.

Of course, there are many more new features in VB 6.0. It would be enough to say that VB 6.0 has firmly established the VB language as a programming tool for the future.

## 2.2 VISUAL BASICS

Visual Basic (VBasic or simply VB) is a high level programming language that allows you to create any type of program for any purpose. The program may range from a simple application to a sophisticated one, in any field such as Education, Accounting, Business, Law, Research and Science.

In addition to this, Visual Basic provides a graphical environment consisting of a set of tools that guides you in creating applications with less programming effort. An application that may require several days to code in another programming language can be done in a few hours with Visual Basic.

Besides, while working in Visual Basic your sense of creativity is aroused. As usual, it allows you to design your work and view the result. However, if you are not satisfied with the outcome, you can edit and modify the design until it matches your requirement. This is indeed a very interesting feature of Visual Basic that will help you a lot while attempting the design stage of your project.

It is essential for you to know that Visual Basic comes in three versions:

- Learning Edition
- Professional Edition, and
- Enterprise Edition.

The last two editions are appropriate for professional programmers whereas the Learning Edition is good for beginners. So, while installing Visual Basic 6.0 on your PC, choose the Learning Edition.

### 2.2.1 Evolution of Visual Basic

VB 1.0 was introduced in 1991. The drag and drop design for creating the user interface is derived from a prototype form generator developed by Alan Cooper and his company called Tripod. Microsoft contracted with Cooper and his associates to develop Tripod into a programmable form system for Windows 3.0, under the code name Ruby (no relation to the Ruby programming language).

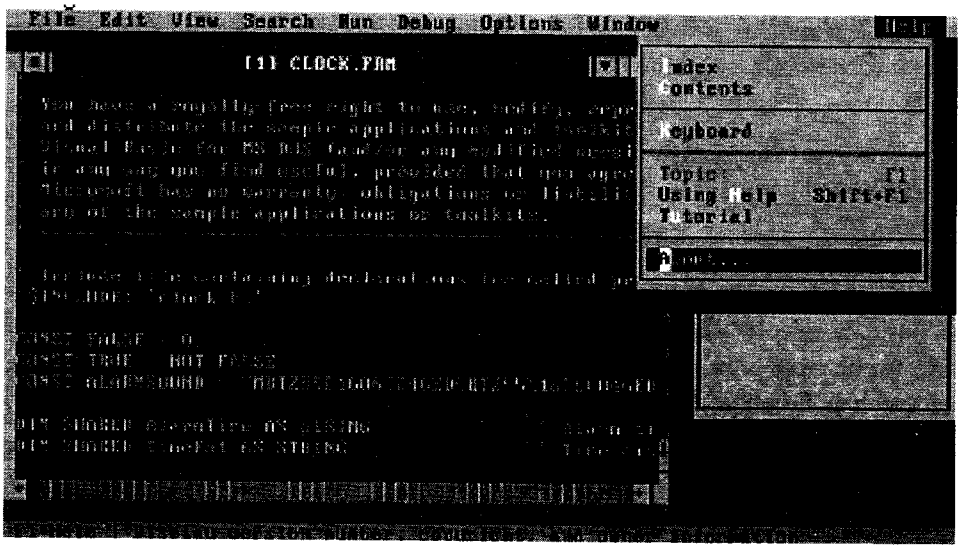
Tripod did not include a programming language at all. Microsoft decided to combine Ruby with the Basic language to create Visual Basic.

The Ruby interface generator provided the "visual" part of Visual Basic and this was combined with the "EB" Embedded BASIC engine designed for Microsoft's abandoned "Omega" database system. Ruby also provided the ability to load dynamic link libraries containing additional controls (then called "gizmos"), which later became the VBX interface.

*Timeline of Visual Basic (VB1 to VB6)*

Project 'Thunder' was initiated.

Visual Basic 1.0 (May 1991) was released for Windows at the Comdex/Windows World trade show in Atlanta, Georgia.




*Visual Basic for MS-DOS*

- Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, QuickBASIC and BASIC Professional Development System. The interface used the "COW" (Character Oriented Windows) interface, using extended ASCII characters to simulate the appearance of a GUI.
- Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved. Notably, forms became instantiable objects, thus laying the foundational concepts of class modules as were later offered in VB4.
- Visual Basic 3.0 was released in the summer of 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.
- Basic 4.0 (August 1995) was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to write non-GUI classes in Visual Basic. Incompatibilities between different releases of VB4 caused installation and operation problems. 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 ActiveX controls.
- With version 5.0 (February 1997), Microsoft released Visual Basic exclusively for 32-bit versions of Windows. Programmers who preferred to write 16-bit programs were able to import programs written in Visual Basic 4.0 to Visual Basic 5.0, and Visual Basic 5.0 programs can easily be converted with Visual Basic 4.0. Visual Basic 5.0 also introduced the ability to create custom user controls, as well as the ability to compile to native Windows executable code, speeding up calculation-intensive code execution. A free, downloadable Control Creation Edition was also released for creation of ActiveX controls. It was also used as an introductory form of Visual Basic: a regular .exe project could be created and run in the IDE, but not compiled.
- Visual Basic 6.0 (Mid 1998) improved in a number of areas including the ability to create web-based applications. VB6 has entered Microsoft's "non-supported phase" as of March 2008.

- Mainstream Support for Microsoft Visual Basic 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. Ironically, around this time (2005), it was exposed that Microsoft's new anti-spyware offering, Microsoft AntiSpyware (part of the GIANT Company Software purchase), was coded in Visual Basic 6.0. Its replacement, Windows Defender, was rewritten as C++ code.

### 2.2.2 Advantage of Visual Basic

There are quite a number of reasons for the enormous success of Visual Basic (VB):

- The structure of the Basic programming language is very simple, particularly as to the executable code.
- VB is not only a language but primarily an integrated, interactive development environment ("IDE").
- The VB-IDE has been highly optimized to support Rapid Application Development ("RAD"). It is particularly easy to develop graphical user interfaces and to connect them to handler functions provided by the application.
- The graphical user interface of the VB-IDE provides intuitively appealing views for the management of the program structure in the large and the various types of entities (classes, modules, procedures, forms, ...).
- VB provides a comprehensive interactive and context-sensitive online help system.
- When editing program texts the "IntelliSense" technology informs you in a little popup window about the types of constructs that may be entered at the current cursor location.
- VB is a component integration language which is attuned to Microsoft's Component Object Model ("COM").
- COM components can be written in different languages and then integrated using VB.
- Interfaces of COM components can be easily called remotely via Distributed COM ("DCOM"), which makes it easy to construct distributed applications.
- COM components can be embedded in / linked to your application's user interface and also in/to stored documents (Object Linking and Embedding "OLE", "Compound Documents").

There is a wealth of readily available COM components for many different purposes.

## 2.3 ANATOMY OF VISUAL BASIC PROGRAM

### 2.3.1 Statements in Visual Basics

Every procedural programming language has five common programming statements. Visual Basic is no exception. So expect to see the following type of statements in just about every VB program which you use or write:

| Statement type | Examples |
|---|---|
| *Declarations* - define the name, type and attributes of all program variables | Dim Num as Integer declares a variable "Num" to be an Integer<br>Dim vals(5) as Double declares an array of 5 Doubles named "vals" |
| *Assignment* - set values for the variables | Num = Num/10 ' after the assignment Num is set to 1/10th of its former value<br>HiString = "Hello " + "World" ' value of HiString is set to "Hello World" |
| *Conditionals* - do operations depending on the value of one or more variables | If Num > 0 then Num = Num * 2 ' Basic logic building block<br>Select Case .... End Case 'Case block simplifies GUI programming |
| *Iterations* - control looping for repeated operations | for i = 1 to 5 'For-loop counts through a precise number of steps while ( val(i) > val(imin) ) 'While loops as long as condition remains True |
| *Subroutines* - calls on functions and subroutines | Private Sub Form_Load() 'A subroutine does not return a value<br>Private Function Digit() ' A function returns a value |
| *Special statements* - used to implement unique features | Set MyObject = Your Object 'Set statement assigns object references<br>Print #FileNum, MyObject.Text 'I/O statements like Print, Input Line |

## 2.3.2 Visual Basic Editing Tools Statement in Visual Basic

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

### *Modules*

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

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

### *Declarations*

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

### *Procedures*

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

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

- A name must begin with a letter.
- May be as much as 255 characters long (but don't forget that somebody has to type the stuff!).
- Must not contain a space or an embedded period or type-declaration characters used to specify a data type; these are ! # % $ & @
- Must not be a reserved word (that is part of the code, like Option, for example)

- The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of First-name use First_name or FirstName.

## 2.3.3 Data Types in Visual Basic

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and more everyday. Similarly in Visual Basic, we have to deal with all sorts of data, some can be mathematically calculated while some are in the form of text or other forms. VB divides data into different types so that it is easier to manage when we need to write the code involving those data.

### *Visual Basic Data Types*

Visual Basic classifies the information mentioned above into two major data types, they are the numeric data types and the non-numeric data types.

### *Numeric Data Types*

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and more. Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others. In Visual Basic, numeric data are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures or data that does not need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, they are also called floating point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve a many decimal points, we can use the decimal data types.

| Type | Storage | Range of Values |
|---|---|---|
| Byte | 1 byte | 0 to 255 |
| Integer | 2 bytes | -32,768 to 32,767 |
| Long | 4 bytes | -2,147,483,648 to 2,147,483,648 |
| Single | 4 bytes | -3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values. |
| Double | 8 bytes | -1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values. |
| Currency | 8 bytes | -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| Decimal | 12 bytes | +/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places). |

### *Non-numeric Data Types*

Non-numeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type.

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

*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.

*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.

*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.

### 2.3.4 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data.

| Suffix | Data Type |
|---|---|
| & | Long |
| ! | Single |
| # | Double |
| @ | Currency |

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

memberName="Turban, John."

TelNumber="1800-900-888-777"

LastDay=#31-Dec-00#

ExpTime=#12:00 am#

### 2.3.5 Operators in Visual Basic

*Arithmetical Operators*

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

*Relational Operators*

| Operators | Description | Example | Result |
|---|---|---|---|
| > | Greater than | 10>8 | True |
| < | Less than | 10<8 | False |
| >= | Greater than or equal to | 20>=10 | True |
| <= | Less than or equal to | 10<=20 | True |
| <> | Not Equal to | 5<>4 | True |
| = | Equal to | 5=7 | False |

*Logical Operators*

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

## 2.3.6 Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In term of VB, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Visual Basic, you have to follow a set of rules:

*Variable Names*

The following are the rules when naming the variables in Visual Basic:

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

| Valid Name | Invalid Name |
|---|---|
| My_Car | My.Car |
| ThisYear | 1NewBoy |
| Long_Name_Can_beUSE | He&HisFather<br>*& is not acceptable |

*Variable Declaration*

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.

There are three ways for a variable to be typed (declared):

(a) Default

(b) Implicit

(c) Explicit

1. If variables are not implicitly or explicitly typed, they are assigned the variant type by default. The variant data type is a special type used by Visual Basic that can contain numeric, string, or date data.
2. To implicitly type a variable, use the corresponding suffix shown in the Data type table. For example

```
TextValue$ = "This is a string"
creates a string variable, while
Amount% = 300
```

Creates an integer variable.

3. There are many advantages to explicitly typing variables. Primarily, we insure all computations are properly done, mistyped variable names are easily spotted, and Visual Basic will take care of insuring consistency in upper and lower case letters Used in variable names. Because of these advantages, and because it is good Programming practice, we will explicitly type all variables.

*Note:* To explicitly type a variable, you must first determine its scope.

*Scope of Variables*

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

- *Global Variable:* Variable accessible anywhere in VB, in all forms that are a part of the project.
- *Module-Level Variable:* Variable that to all procedures on the form in which it is declared.
- *Local Variable:* Variable accessible only in the procedure which it was declared.

*Local Variables*

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

*Dim sum as Integer*

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

### *Static Variables*

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

### *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

Running Total = 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.

### *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.

- Public ans as Integer
- Private Temp as Integer

Declaring a variable with Public keyword makes it available throughout the application even for the modules. At the module-level there is no difference between Dim and Private, but Private is preferred

because it makes the code easier to read. Public variable should not be declared within a procedure. Public variables in different modules can share the same name and they can be differentiated in code. For example, if the public integer variable intY is declared in both Form1 and Module 1 of a project it can be referred as Form1.intY and Module1.intY

*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.

*In Short Description of Variable*

The levels are:

- Procedure level
- Procedure level, static
- Form and module level
- Global level
- Within a procedure, variables are declared using the Dim statement:
  - Dim MyInt as Integer
  - Dim MyDouble as Double
  - Dim MyString, YourString as String

  Procedure level variables declared in this manner do not retain their value once a procedure terminates.
- To make a procedure level variable retain its value upon exiting the procedure, replace the Dim keyword with Static:
  - Static MyInt as Integer
  - Static MyDouble as Double
- Form (module) level variables retain their value and are available to all Procedures within that form (module). Form (module) level variables are declared. In the declarations part of the general object in the form's (module's) code Window. The Dim keyword is used:
  - Dim MyInt as Integer
  - Dim MyDate as Date
- Global level variables retain their value and are available to all procedures within an application. Module level variables are declared in the declarations part of the general object of a module's code window. (It is advisable to keep all global variables in one module.) Use the Global keyword:
  - Global MyInt as Integer
  - Global MyDate as Date

- What happens if you declare a variable with the same name in two or more places? More local variables shadow (are accessed in preference to) less local variables. For example, if a variable MyInt is defined as Global in a module and declared local in a routine MyRoutine, while in MyRoutine, the local value of MyInt is accessed. Outside MyRoutine, the global value of MyInt is accessed.

*Example of Variable Scope:*

**Module1**

| | |
|---|---|
| Global X as intger<br>Form1 | Dim Z as Single<br>Form2 |
| Dim Y as Intger | Dim Z as Single |
| Sub Routine1()<br>Dim A as Double<br>End Sub | Sub Routine2()<br>Dim A as Double.<br>End Sub |
| Sub Routine2()<br>Dim A as Double<br>End Sub | |

Procedure Routine1 has access to X, Y, and A (loses value upon termination)

Procedure Routine2 has access to X, Y, and B (retains value)

Procedure Routine3 has access to X, Z, and C (loses value)

*Module1*

Global X as Integer

*Form1 Form2*

```
Dim Y as Integer Dim Z as Single
Sub Routine1() Sub Routine3()
Dim A as Double Dim C as String
. .
. .
End Sub
Sub Routine2()
Static B as Double
.
.
End Sub
```

Procedure Routine1 has access to X, Y, and A (loses value upon termination)

Procedure Routine2 has access to X, Y, and B (retains value)

Procedure Routine3 has access to X, Z, and C (loses value)

### *Explicit Declaration*

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

The format is as follows:

Dim Variable Name As Data Type

*Example*

- Dim password As String
- Dim yourName As String
- Dim firstnum As Integer
- Dim secondnum As Integer
- Dim total As Integer
- Dim doDate As Date

You may also combine them in one line, separating each variable with a comma, as follows:

Dim password As String, yourName As String, firstnum As Integer,.....

If data type is not specified, VB will automatically declare the variable as a Variant.

For string declaration, there are two possible formats, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as previous example. However, for the fixed-length string, you have to use the format as shown below:

Dim VariableName as String * n, where n defines the number of characters the string can hold.

*Example*

Dim yourName as String * 10

Your Name can holds no more than 10 Characters

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 = intcont + 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 mityped 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.

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

### 2.3.7 Constants

Constants are always declared using the keyword CONST. You give the constant a name, data type and value. Once a value is declared a constant, it can never be changed again in the program. Trying to change a constant will cause an error. The rules for variables also apply to constants.

Syntax:

```
DIM <constant name> as <datatype>=value
```

*Example*

```
DIM sngPI as Single = 3.14
DIM curTaxRate as Currency = .07
```

***Named Constants:*** These are constants that you name yourself with the CONST keyword. Named constants can come in the form as numeric constants and string constants.

***Numeric Constants:*** Constants that can contain only the digits 0-9, a decimal pt and sign.

***String Constants:*** Constants that can contain letters, digits, and special characters such as @ # $ % ^ & * . String constants must be enclosed in double quotes.

***Instrinct Constants:*** System-Defined constants that are built into vb.

*Examples of Instrinct Constants*

- vbRed
- vbGreen
- vbBlue
- Checked
- vbYellow

### *Assigning /Working Values to Variables*

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is:

Variable = Expression

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and more. The following are some examples:

```
firstNumber=100
secondNumber=firstNumber-99
username="John"
userpass.Text=
userName="JohnLyan"
userpass.Text = password
Label1.Visible = True
Command1.Visible = false
Label4.Caption = textbox1.Text
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber+ThirdNumber
```

## 2.3.8 Input Boxes

A function that will display a message and allow the user to enter information in a text box. In the Input box you can display a message called a prompt, which will help the user decide what information he needs to enter in the text box.

The input box will have a txt box with the prompt above it and two command buttons, OK and CANCEL. The OK will accept whatever input the user enters and place it in the variable on the left hand side of the equals sign. The CANCEL button will ignore any input entered by the user and return the user back to the form that is currently open Input Boxes are often used when one wants to retrieve records from files

```
VariableName = InputBox("Prompt","Title")
```

### *Example*

```
StrName = InputBox(&quot;Enter your name&quot;,
&quot;Sponge Bob&quot;)<br>
StrCareer=InputBox(&quot;Enter your workplace&quot; &quot; Nickelodeon&quot;)
```

The prompt must be enclosed in quotes. The title must also be in quotes and will appear in the Title Bar of the Input Box. If no title is entered, the title of the project will appear in the Title Bar.

Input Boxes can appear in the following places:

Form_Load

Command Buttons

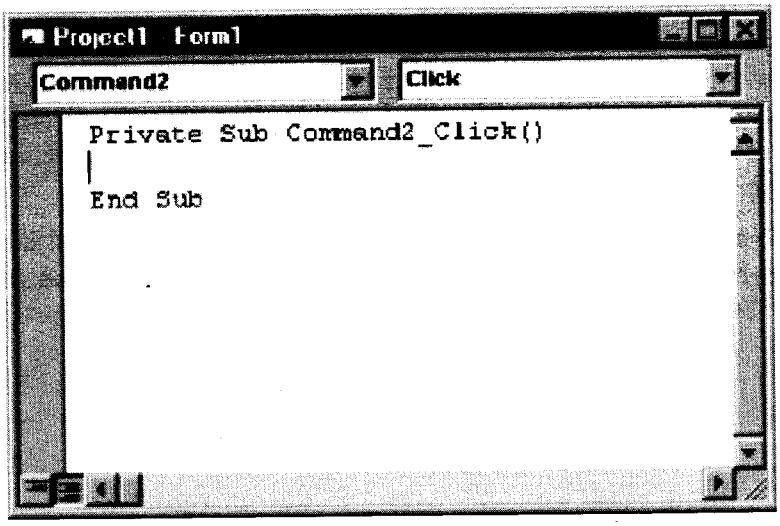
Option Buttons

Check Boxes

## 2.4 THE CODE WINDOW

1. Double-click the End command button on the form.

   The Code window appears, as follows:




   If the window is smaller than the one shown above, resize it with the mouse. (The exact size is not that important because the Code window includes scroll bars that you can use to examine long program statements.)

   Inside the Code window are program statements that mark the beginning and the end of this particular Visual Basic subroutine, or event procedure, a block of code associated with a particular object in the interface:

```
Private Sub Command2_Click()
End Sub
```

   The body of a procedure always fits between these lines and is executed whenever a user activates the interface element associated with the procedure. In this case, the event is a mouse click, but as you'll see later in the book, it could also be an event of a different type.

2. Type End, and press the Down arrow key.

   As you type the statement, the letters appear in black type in the Code window. When you press the Down arrow key (you could also press Enter or simply click a different line), the program statement turns blue, indicating that Visual Basic recognizes it as a valid statement, or keyword, in the program.

   You use the program statement End to stop your program and remove it from the screen. The Visual Basic programming system contains several hundred unique keywords such as this, complete with their associated operators and symbols. The spelling of and spacing between these

items are critical to writing program code that will be accurately recognized by the Visual Basic compiler.

The End statement stops the execution of a program.

Another name for the exact spelling, order, and spacing of keywords in a program is statement syntax.

3. Move the cursor to the beginning of the line with the End statement in it, and press the Spacebar four times.

   The indent moves the End statement four spaces to the right to set the statement apart from the Private Sub and End Sub statements. This indenting scheme is one of the programming conventions you'll use throughout this book to keep your programs clear and readable. The group of conventions regarding how program code is organized in a program is often referred to as program style.

   Now that you've written the code associated with the End button, you'll write code for the Spin button. These programming statements will be a little more extensive and will give you a chance to learn more about program syntax and style. You'll study each of the program statements later in the book, so you don't need to know everything about them now. Just focus on the general structure of the program code and on typing the program statements exactly as they are printed. (Visual Basic is fussy about spelling and the order in which keywords and operators appear.)

## 2.5 PROPERTY SETTING

With some properties you can set and return their values - these are called read-write properties. With some others you can only return their values - these are read-only properties.

We have already met examples of setting a property, e.g.

```
Worksheets("Sheet1").Range("A1").Value = 42
Worksheets("Sheet1").Range("A1").Value = _
Worksheets("sheeet2").Range("B2").Value
To get the value property of cell A1 in sheet1 we would use
myValue = Worksheets("sheet1").Range("A1").Value
```

There are some properties and methods that are unique to collections. The Count property is one. It returns the number of elements in a collection. It is useful if you want to loop over the elements of the collection (though the For Each ... Next loop is preferable). The following example uses the Count property to loop over the worksheets in the active workbook, hiding alternate ones:

```
Sub HideEveryOtherSheet()
For i = 1 To Worksheets.Count
If i Mod 2 = 0 Then
Worksheets(i).Visible = False
End If over collections
Next i
End Sub
```

## 2.6 STRING

Visual Basic has a rich set of operators and in-built string functions. A string expression is one which is produces a string value. A number of interesting operations can be performed on string type data values.

Among the operators that can be used on string type values are concatenation. Concatenation operator (i.e., + or &) takes two string type operands and produces a string that is concatenation of the two. Consider the following for example,

"India " + "is a great country"

This expression evaluates to a single string as shown below:

"India is a great country"

However, note that all the arithmetical operations are not applicable on the string type values. You cannot for example, use subtraction, multiplication or division operations on the string type values.

### 2.6.1 Library Functions

Fortunately, Visual Basic provides a range of string functions that programmers may use to their advantage in their programs. A complete list of all the functions may be found in the Standard References. However, a synopsis of the same is presented hereunder.

*Str(Numeric)*

This function takes a single numeric operand and produces a string of that value. For example,

Str(9.201) → "9.201".

*Val(String)*

This function does just the opposite. It takes a string type operand and produces its numeric value if the input represents a number. For example,

Val("234.12") → 234.12

*Left(String, long)*

This function takes two operands - a string and an long and returns long many characters from its left side. For example,

Left("Computer", 3) → "Com"

*Right(String, long)*

This function takes two operands - a string and a long and returns *long* many characters from its right side. For example,

Right("Computer", 3) → "ter"

*Mid(String, integer, long)*

This function takes three operands - a string and an two integers. It returns second integer many characters starting from first integer position. For example,

Mid("Computer", 3, 2) → "mp"

Clearly, "mp" is 2 characters (second integer) from the third position (first integer).

***Space(long)***

This function returns a string having as many spaces as is specified by the long type operand. Thus,

Space(10) → "  "

***Ltrim(String)***

This function takes a string type and returns the same by removing all the leading spaces from the left. For example,

Ltrim("  Computer ") → "Computer "

***Rtrim(String)***

This function takes a string type and returns the same by removing all the leading spaces from the right. For example,

Rtrim("  Computer ") → "  Computer"

***Trim(String)***

This function takes a string type and returns the same by removing all the leading spaces from both sides. For example,

Trim("  Computer ") → "Computer"

***String(Long, Character)***

This function returns a string having Long many Characters. For example,

String(10, "*") → "**********"

***StrReverse(String)***

This function returns a string by reversing the input string. For example,

StrReverse("Computer") → "retupmoC"

***Len(String)***

This function returns the number of characters in the string. For example,

Len("Computer") → 7

## 2.7 CONTROL STATEMENTS/MAKING DECISION

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

### *If...Then selection structure*

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

Syntax of the If...Then selection

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

e.g.: If average > 75 Then

```
txtGrade.Text = "A"
End If
```

### *If...Then...Else selection structure*

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

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

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

e.g.: If average > 50 Then

```
txtGrade.Text = "Pass"
Else
txtGrade.Text = "Fail"
End If
```

### *Nested If...Then...Else selection structure*

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

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

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

### *Method 1*

```
If < condition 1 > Then
statements
ElseIf < condition 2 > Then
statements
ElseIf < condition 3 > Then
statements
```

```
Else
statements
End If
```

***Method 2***

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

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

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

***Select...Case selection structure***

If the selection involves more than two alternatives, you can use nested If statements but this becomes complicated and leads to hard-to-read code. It is better to use Case statements. Here is the syntax for multiple selection through Case statements.

Syntax of the Select...Case selection structure

```
Select Case Expression
Case value
[One or more VB statements]
```

```
Case value
[One or more VB statements]
Case value
[One or more VB statements]
Case Else
[One or more VB statements]
End Select
```

***Example***

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

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

***Example***

```
Select Case intAge
Case Is < 6
lblTitle.Caption = "Preschool"
Case 6 To 11
lblTitle.Caption = "Primary School"
Case 12 To 18
lblTitle.Caption = "Secondary School"
Case Else
lblTitle.Caption = "Adult"
End Select
```

**Check Your Progress**

1. What are the Instrinct Constants?
2. State whether the following statements are true or false:
   (i) Static variables are reinitialized each time.
   (ii) The Operator & is use as AND operator.
3. Fill in the blanks:
   (i) Non-numeric data types are data that cannot be manipulated ..... using standard arithmetic operators.
   (ii) ..... Variable accessible anywhere in VB.

## 2.8 LET US SUM UP

As we face many types of data that come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and more everyday. In the same way we in Visual Basic, we have to deal with all sorts of data, some can be mathematically calculated while some are in the form of text or other forms. VB divides data into different types so that it is easier to manage when we need to write the code involving those data. In this lesson we also learn that how to create Visual Basic code that can accept input from the user and display the output without controlling the program flow. In this lesson, we learned that how we create VB code that can make decision when it process input from the user, and control the program flow in the process. As we know that Decision making process is an important part of programming because it can help to solve practical problems intelligently so that it provide useful output or feedback to the user. For example, we can write a program that can ask the computer to perform certain task until a certain condition is met.

Also we learned the another way to control the program flow, that was, the Select Case control structure.

The difference is that the Select Case control structure basically only make decision on one expression or dimension (for example the examination grade) while the If ...ElseIf statement control structure may evaluate only one expression, each If....ElseIf statement may also compute entirely different dimensions. Select Case is preferred when there exist many different conditions.

## 2.9 KEYWORDS

***Iteration:*** It refers to the meaning of loop in visual basic.

***Double:*** This is a type of Datatype.

***COBOL:*** This is a type of Procedural language.

***Modules:*** Code in Visual Basic is stored in the form of modules.

***Anatomy:*** Means structure or framework of any thing.

***Non-procedural_Languages:*** Object-oriented programming languages that are Event-driven.

## 2.10 QUESTIONS FOR DISCUSSION

1. Explain the meaning of the term:
   (a) Procedures
   (b) Variant
   (c) Static Variables
2. What is the scope of variables?
3. What is constants? What is the type of constants?

**Check Your Progress: Model Answers**

1. Instrinct Constants are the System-Defined constants that are built into VB Examples of Instrinct Constants vbRed.
2. (i) False (ii) False
3. (i) mathematically (ii) Global Variable

## 2.11 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# UNIT II

LESSON

# 3

# WORKING WITH OBJECT AT RUN TIME

CONTENTS



## 3.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of projects with multiple forms
- Discuss how to display information
- Identify and explain the printer object

## 3.1 INTRODUCTION

You should be familiar with VB's event-driven programming style by now. However I'll explain it again. When you're doing general VB programming, your thoughts should go in this pattern: "If the user does this, what should happen? How would I make it happen?" Then you would write the program to fit the answers to these questions. That is event driven programming. You program according to what events the user would trigger. Dragging a picture, clicking a button, and typing a word are all events. You, being the brave coder that you are, would ask: "Why do I have to think like that?"

Well here's an alternative way for you to think: Object Oriented Programming (OOP). In the world of OOP, you break a problem down into small parts and solve them individually. If you are to program in an object oriented style, you would think of every variable or functions as a property of an object, and everything would seem like an object to you. OOP is hard to explain so you'll have to experience it in the following subsections.

## 3.2 VISUAL BASIC PROJECT

Visual basic project basically consists of three parts – the user interface, the processing of information, and the storage of information.

The user interface is the part of the program that your users see and with which they interact. This user interface is composed of the screens you design by using Visual Basic's forms and controls.

The processing of information is done by the set of statements written to add the functionality to the object.

The storage for information is generally any backend Database, which is connected to frontend by any of the standard Data Control or through Designers.

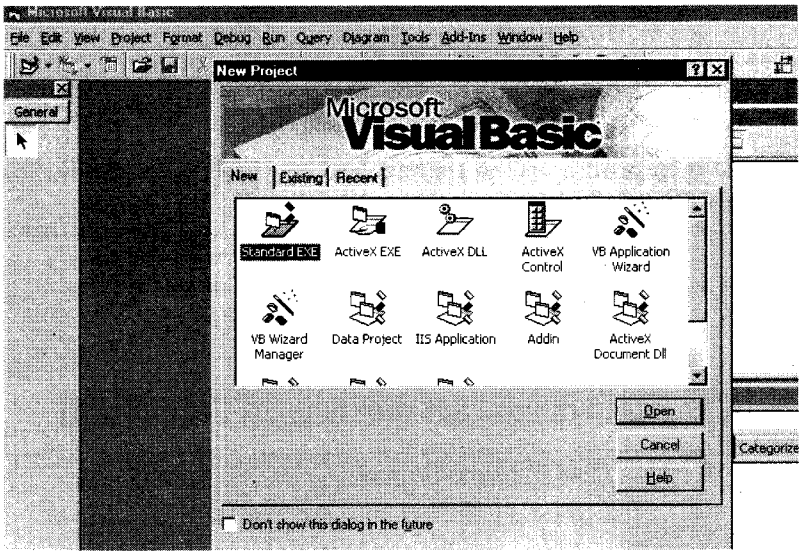
Following are the Modules which are available in Visual Basic project to perform the above three tasks:

1. *Form Module:* It contains the graphic elements of the VB application along with the instruction.
2. *General Module:* It contains general-purpose instructions not pertaining to anything graphic on-screen.
3. *Class Module:* It contains the defining characteristics of a class, including its properties and methods.
4. *Global Module:* It contains declaration and procedures.
5. *Resource Files:* It allows you to collect all of the texts and bitmaps for an application in one place.

When you start VB for the first time project wizard will be opened and following project templates will be displayed

1. *Standard EXE:* It is a typical Application.
2. *ActiveX EXE, ActiveX DLL:* Available with Professional Edition. ActiveX components are basic code-building elements that don't have a visible interface and that can add functionality to your applications.
3. *ActiveX Control:* Used to develop your own ActiveX controls. It is a basic element of the user interface. If the ActiveX control that comes with Visual Basic don't provide the required functionality then custom ActiveX controls can be build.
4. *ActiveX Document DLL, ActiveX Document EXE:* ActiveX documents are in essence Visual Basic applications that can run in the environment of a container that supports hyperlinking.
5. *VB Application Wizard, VB Wizard Manager:* The Application Wizard takes through the steps of setting up the skeleton of a new application like a template in any MS-office component where basic structure is readily available and changes can be made depending on the need. Whereas VB Wizard Manager helps in building our own Wizard which is a sequence of windows that collects information from user required for building an application.
6. *Data Project:* This is a feature of enterprise edition that automatically adds the controls that are used in accessing databases to the Toolbox.

7. ***DHTML Application:*** Allows to build Dynamic HTML pages that can be displayed in the browser's window on a client computer. It helps user in viewing the data lying with any RDBMS package at the backend and can also edit that and can do the queries also through browser itself.
8. ***IIS Application:*** Allows to build applications that run on the Web server and interact with clients over the Internet with Internet Information Server.
9. ***Add-In:*** Own Add-Ins can be created for the Visual basic IDE. Add-Ins are special commands that can be added to Visual Basic menus.
10. ***VB Enterprise Edition Control:*** It simply creates a new Standard EXE project and loads all the tools of the Enterprise Edition.
11. ***Steps for building a Visual Basic Application:*** After designing the application, for building an application in Visual Basic following tips are to be taken:
    (a) *Draw the Interface:* Create the form and the various objects on the form using toolbox.
    (b) *Set the Properties:* Set the properties of the various objects on the form with the help of properties window.
    (c) *Write the events code:* In the code window, write the code associated with each object on the form and then run the application.




*Few Examples to start with Visual Basic*

How to Start a project

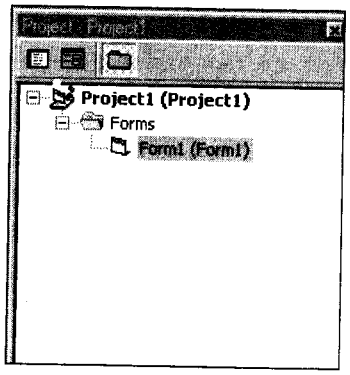
_Start the visual basic by clicking Start> >Programs> >Microsoft Visual Studio 6.0> >Microsoft Visual Basic

Choose Standard EXE from the available screen

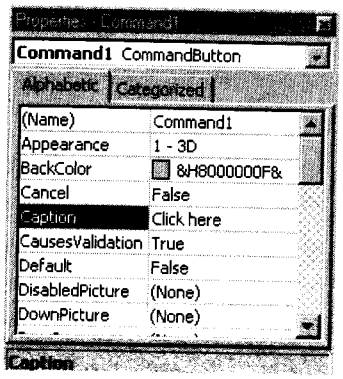
Click Open.

A blank form will appear on the screen with Project Explorer Window, Properties Window and Tool Box.

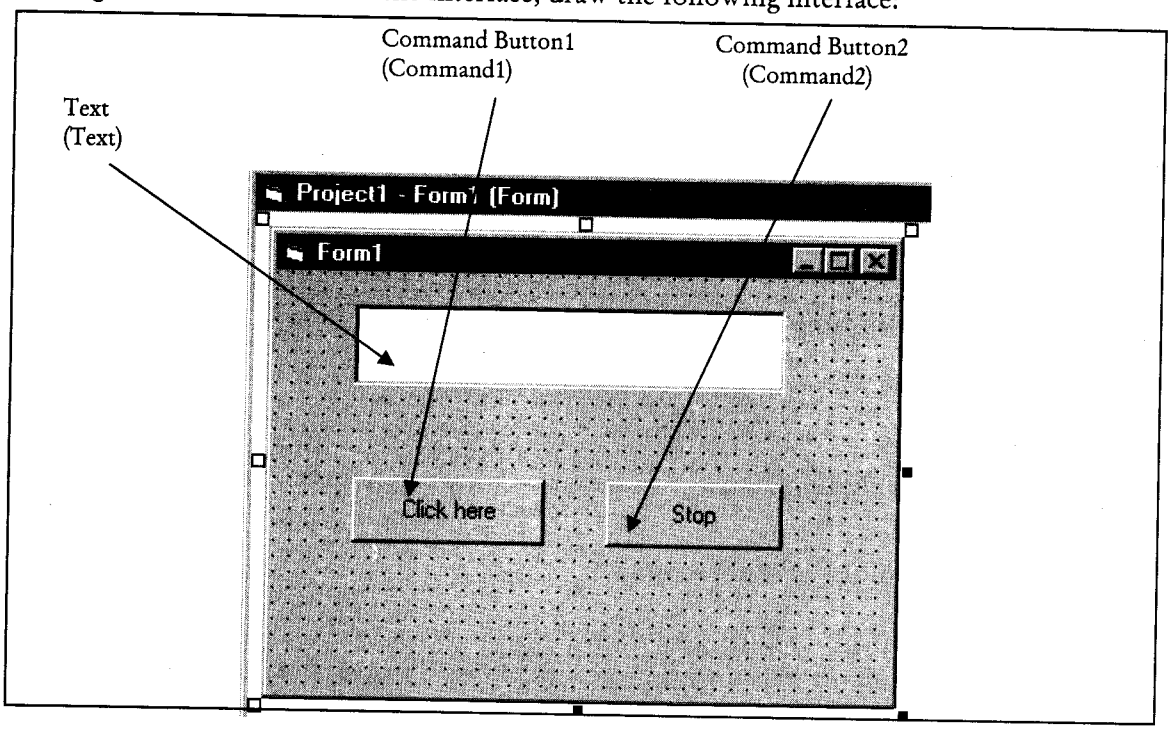
Project Explorer Window




Properties Windows




***Example 1:*** This example is to print a text in the text box by clicking on the button. With the help of Tool Box given on the left side of the interface, draw the following interface:



Set the properties of the objects on the form as given in the table below:

| Object | Name | Property | Value |
|---|---|---|---|
| Text Box | Text1 | Text | <Blank> |
| Command Button | Command1 | Caption | Click here |
| Command Button | Command2 | Caption | Stop |

_Double click First Command Button and write the following lines of code

```
Private Sub Command1_Click()
Text1.Text = "Welcome to Visual Basic"
End Sub
```

_Click View Object to come back at the form

_Double click Second Command Button and write the following lines of code

```
Private Sub Command2_Click()
End
End Sub
```

_Run the project by clicking Run > > Start or Press F5.

_Save the project by clicking File > > Save Project.

_Specify the folder in which all the files like form, reports etc. related with the project along with the project name are to be saved.

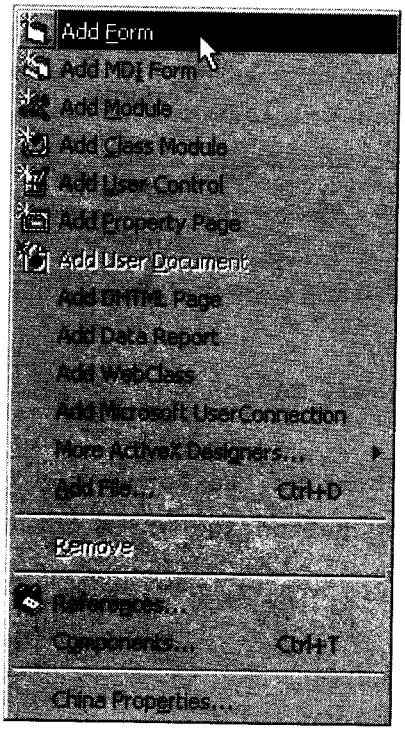
### 3.2.1 Projects with Multiple Forms

We Start a project named the China Shop consisted of a Visual Basic Project with a single form.

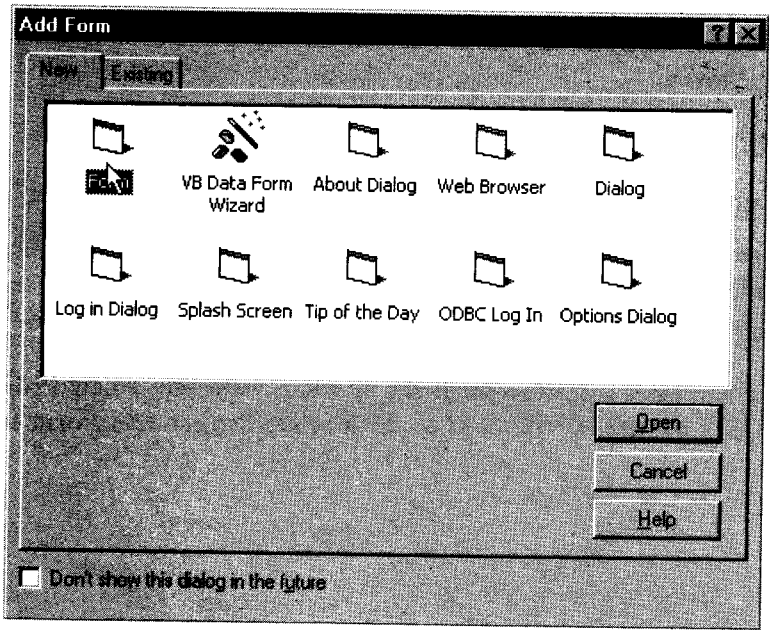
Most real world projects will have more than one form. In this we add second form to the China Shop Project which we will use to display what I call a 'Help About'.

*Step 1:* Add a New Form to the Project.

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



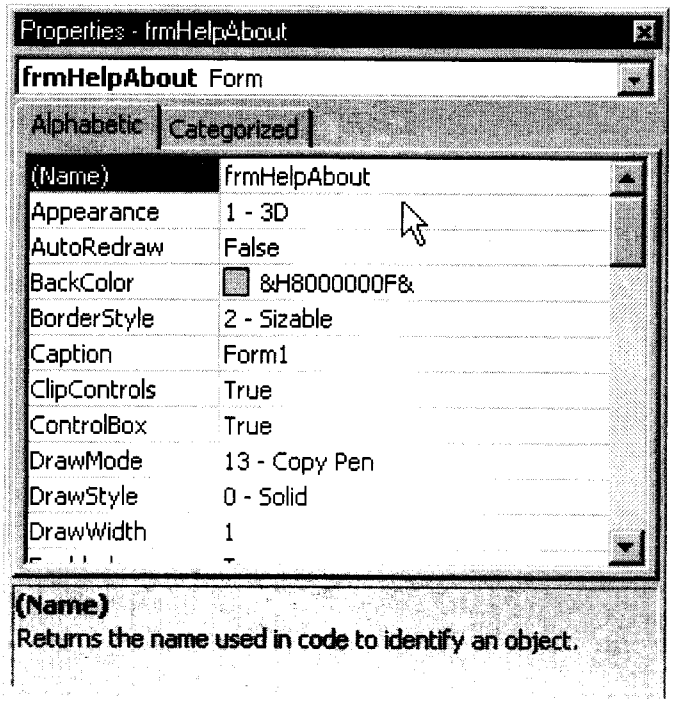

The following screen shot will appear....




At this point, we just need to make sure that the 'New' tab is selected, and then either double-click on 'Form' or select 'Form' and click on the 'Open' Button. Visual Basic will then display a new form for us in the IDE, with a Caption reading 'Form1'.

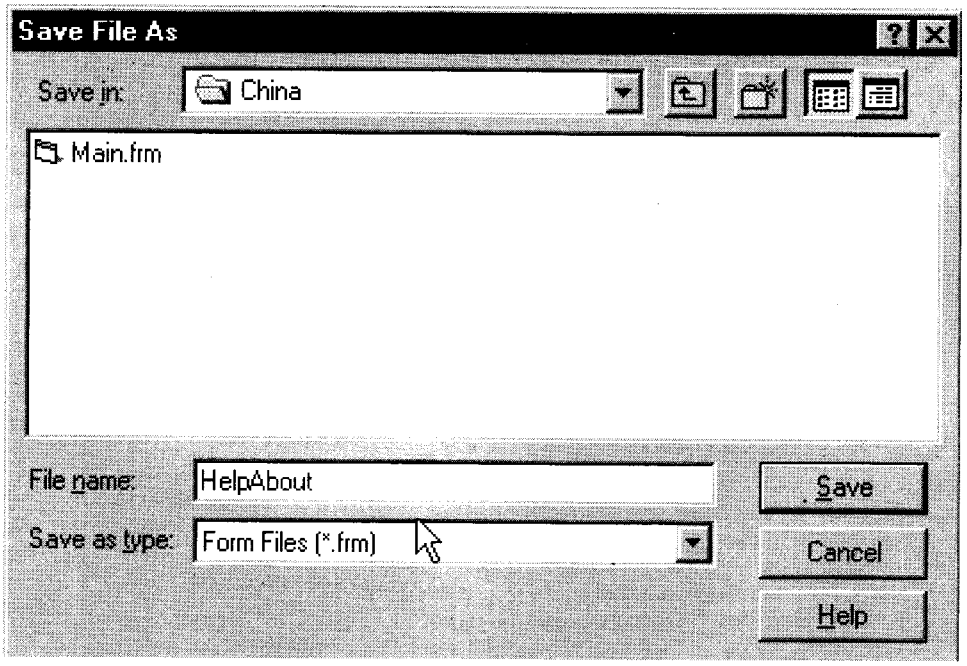
We now have two forms loaded into the IDE. The smart thing to do now is to save the form, but before we do that, we should give it a meaningful name. To do that, select the form in the IDE by clicking on it with the mouse, and then bring up its Properties Window.

Find the Name Property, and change it from Form1 to frmHelpAbout.

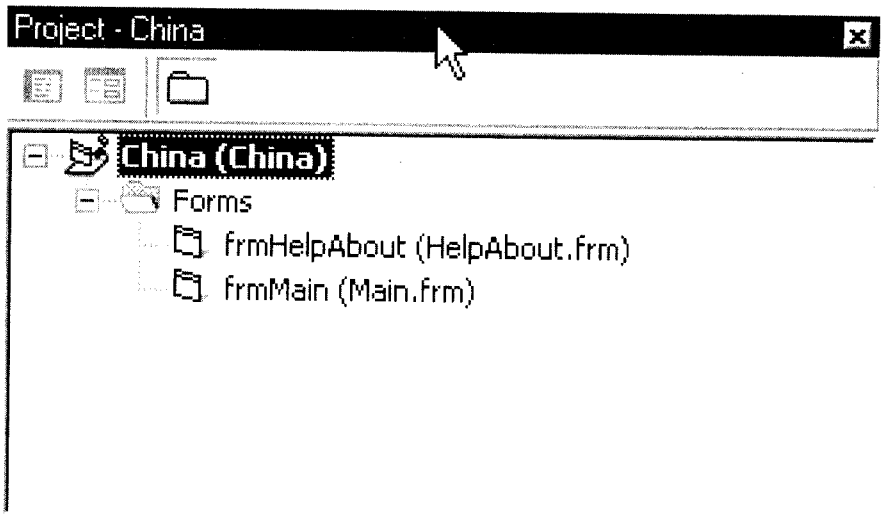



Now click on the Save button on the Toolbar. Visual Basic will immediately prompt you to save the new form with a disk file name of 'frmHelpAbout.'

We can save the form with that name, or change it if we wish. My personal preference is to drop the 'frm' from the disk file name, and save it as HelpAbout. Let's do that by clicking on the Save button, also ensuring that the form will be saved in the correct folder or directory (\VBFILES\CHINA).



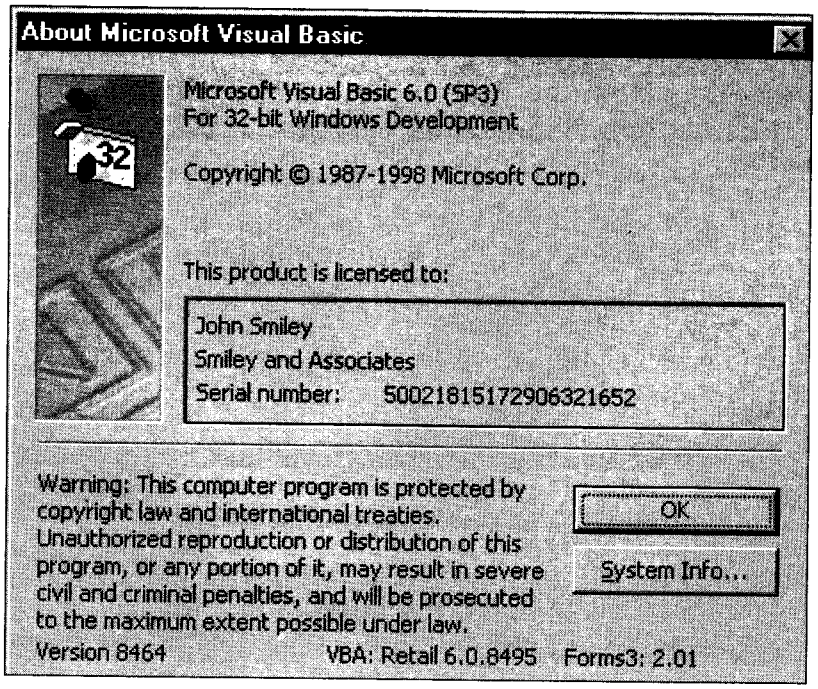
Once the form has been saved, if we bring up the Visual Basic Project Explorer Window, we should now see two forms - Main (with a Name Property of frmMain) and HelpAbout (with a Name Property of frmHelpAbout).




*Step 2:* Design the New Form

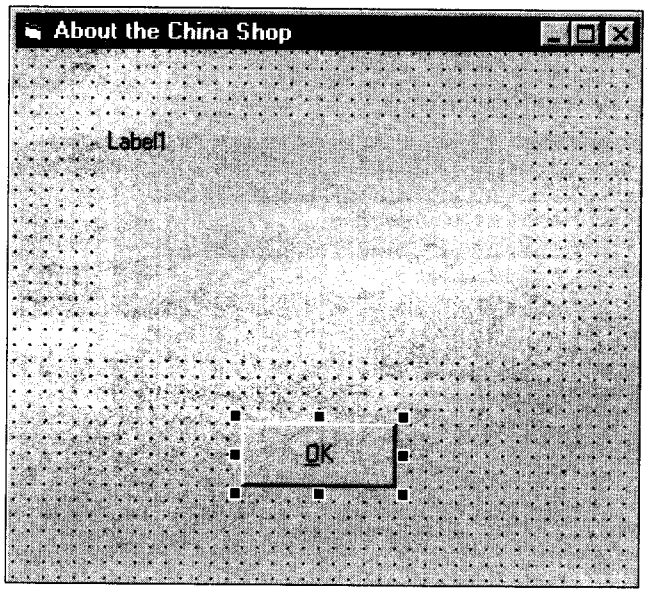
Our project now has two forms - at this point, we need to design the second form, and then we'll add code to display it from the Main form of the China Shop Project.

A Help-About form needn't be too fancy - basically, it should say something about the person who wrote it, and if you're smart, you'll include some contact information in the event that the person viewing it wants to hire you for some work! You can always use the Microsoft Help-About forms as a sample For instance, here's the Help-About form for my copy of Visual Basic 6.



As you can see, the Visual Basic Help-About Form contains information about the program, and also two buttons - one is a System Information button (more on that in a future Web article) and one is an OK button. We need to include an OK button as we need a way to close the form when the user is done viewing it!

Let's resize the frmHelpAbout form now, and add a single Label control and a Command Button. We'll change the name of the Command Button to cmdOK, and change its captionProperty to OK. Finally, change the name of the Label to lblTootYourHorn. Finally, changethe Caption Property of the form to About the China Shop. If you are following along withme, your form should look similar to this.




You may be wondering about all that great information I told you we would be placing on the form. The easiest way to do that is through code - which is our next step.

*Step 3:* Write Code for the New Form

Our next step is to write some code so that when the Help-About Form is displayed (more on how to do that in Step 4), the Caption of the label control displays the information that we wish to display about us (and the program). We will also need to write code that will permit the user to close the form when he or she clicks on the OK button.

Let's start with the Caption of the label - we'll place that code in the Load Event Procedure of the form. Here's the code

```
Private Sub Form_Load()
lblTootYourHorn.Caption = "This program written by John Smiley" & _
vbCrLf & vbCrLf & "Email: johnsmiley@johnsmiley.com" & _
vbCrLf & vbCrLf & "Web Site: http://www.johnsmiley.com"
End Sub
```

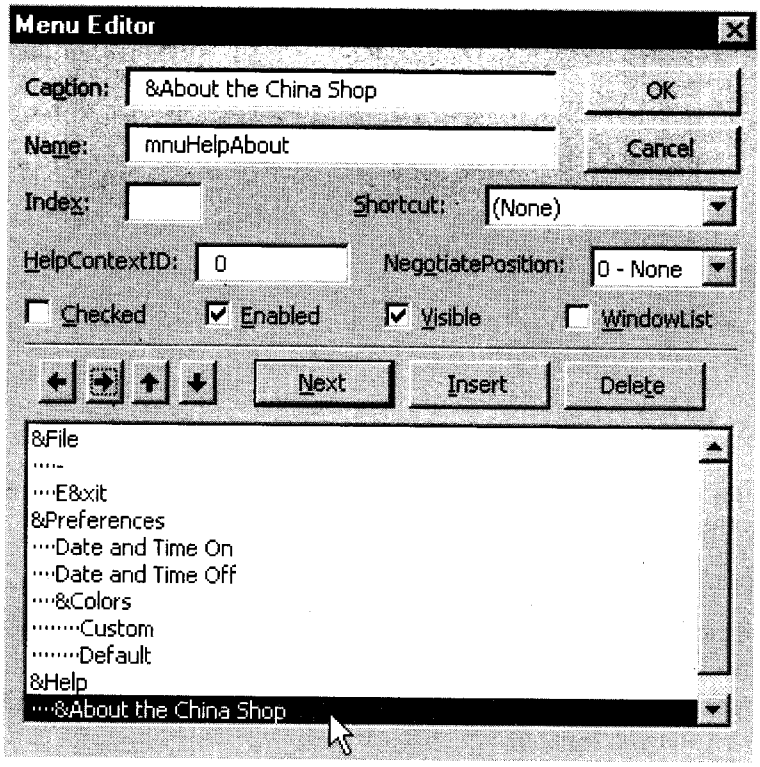
All we're doing here is displaying some information about the person who wrote the program, and providing an email address and a website. VbCrLF is the Visual Basic Intrinsic Constant for a Carriage Return and Line Feed, which enables us to separate those three pieces of information with blank lines.

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

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

Simple enough - Unload Me (where Me is just a shortcut reference to the Current Form) will unload the Help-About Form. And the Microsoft recommendation is to set the name of the form you are unloading (with the exception of your startup form) to Nothing - this will free RAM of the remnants of the form and its code. Now we need to write the code to display the Help About Form.

***Step 4:*** Write Code to display the New Form




What we'll do here is create a Menu entry to display the Help About Form. Not surprisingly, we'll use the Menu Editor to create a menu item called Help, with a submenu item called About. The Menu structure, as seen from the Visual Basic Menu Editor, should look like this...

We'll place the code necessary to display the Help-About form in the Click Event procedure of mnuHelpAbout. Many beginners I know have been able to get this far on their own - but you're about to see the mistake they make - typically, beginners will read about the Load statement in Visual Basic Help and figure that this statement will do the trick...

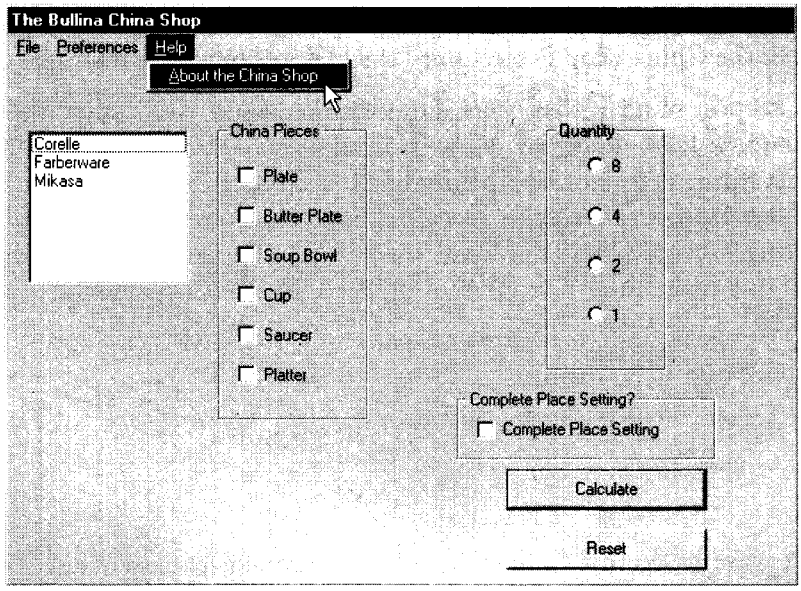
```
Private Sub mnuHelpAbout_Click()
Load frmHelpAbout
End Sub
```

Now before you go off and execute the modified China Shop program, expecting to see the new Help About form displayed when you click on Help-About the China Shop, I should tell you that there's a problem with this code.

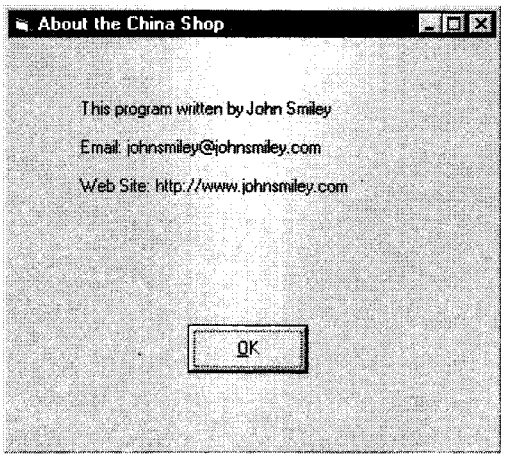
The problem with the code is that while it loads the Help-About form, it doesn't display it! Merely loading a form doesn't make it visible - by default, a loaded form is not visible - it must explicitly be made visible. There are two ways to do that - either set the Visible Property of the form you wish to load to True, or execute its Show Method. As it turns out, the quickest and most efficient way to both load and make the form visible is to just execute its Show Method - which will both load the form and make it visible. Therefore, this is the code to use.

```
Private Sub mnuHelpAbout_Click()
frmHelpAbout.Show
End Sub
```

Now if you click on the Help-About the China Shop menu item...




the Help-About form will be displayed...



Let's take some time to admire our work, and then click on the OK button. The form should then disappear Some nuances...

I should make you aware of some nuances of multiform projects before finishing this month's article. First, some forms (particularly Help-About forms) should be displayed as modal forms. A modal form is one that prevents further processing within the application until the form is closed. In terms of the Help-About form, that means that while the form is displayed, the user cannot interact with the rest of the China Shop project. Choosing to display a form as a modal form is a good idea if the form is meant to be an attention getter - one that you want the user to interact with before doing anything else. Displaying a form as a modal form is easy enough - just use the vbModal argument of the Show Method like this...

```
Private Sub mnuHelpAbout_Click()
frmHelpAbout.Show vbModal
End Sub
```

Now if you run the program, and click on the Help-About form, you'll see that you can't interact with the main form of the China Shop Project until the Help-About form is closed.

Another nuance to keep in mind is that there are occasions when you display the second form that you want to make the first form disappear. Now that can mean one of two things. First, you are done working with the first form, and you won't be needing it again. An example of such a form would be a Login form, which permits secure access to an application. Once the user has been authenticated, the Login form is no longer needed, and the main form of the application is displayed. To code this scenario, you would just unload the first form using the Unload Me statement, and then display the second form. Like this...

```
Private Sub mnuHelpAbout_Click()
Unload Me
frmHelpAbout.Show vbModal
End Sub
```

Obviously, we don't want to do that in the China Shop program, because we still want access to the main form of the China Shop available when the user is done reading the Help-About form.

Beginners, at this point, will sometimes make the mistake of unloading the first form, and then re-loading it from the OK button of the Help-About form - like this...

```
Private Sub cmdOK_Click()
Unload Me
frmMain.Show
Set frmAbout = Nothing
End Sub
```

In general, this technique is not a good one.

Loading forms in Visual Basic is a strain on the PC, so if you want to make the first form invisible while the second is displayed, and then redisplay it later, in general, it's a better idea to Hide the first form instead of unloading it - like this...

```
Private Sub mnuHelpAbout_Click()
Me.Hide
frmHelpAbout.Show vbModal
End Sub
```

Then, when you are done with the Help-About form, use this code to Show the hidden main form...

```
Private Sub cmdOK_Click()
Unload Me
frmMain.Show
Set frmAbout = Nothing
End Sub
```

The main form will instantly appear without having to be re-loaded.

Summary Now that you know how to add a second form to your project, adding additional forms shouldn't be a problem.

## 3.3 PRINTER OBJECT

The Visual Basic 6.0 Printer object is replaced by the PrintDocument component in Visual Basic 2005. The behavior of the two is considerably different, but in most cases the functionality can be duplicated.

### 3.3.1 Conceptual Differences

In Visual Basic 6.0, printing is accomplished by creating a Printer object and using graphics methods to draw text and graphics onto a virtual page. Properties and methods are used to define printer attributes such as DeviceName, PrintQuality or Copies; other properties such as Orientation and PaperSize define attributes of the page itself. The EndDoc method sends the output to the default printer for the application as defined in the Printers collection.

In Visual Basic 2005, the Printer object no longer exists. Instead, you use a PrintDocument component to define the graphics and text, a PrinterSettings object to define printer attributes, and a PageSettings class to define page attributes.

Printing is no longer tied to a specific device, and the concept of a default printer for an application is no longer valid. Instead the PrintPage method of the PrintDocument component can be used to print to any device, and the default printer is system-wide. The PrintDialog, PrintPreviewDialog, and PageSetupDialog components allow you to let the user select a printer and print options at run time.

*ColorMode Property*

In Visual Basic 6.0, the ColorMode property controls whether output is printed in monochrome on a color printer.

In Visual Basic 2005, it is now up to the printer to expose this as an advanced option. The SupportsColor property of the PrinterSettings class can be used to determine the color capabilities of a printer.

*DriverName Property*

In Visual Basic 6.0, the DriverName property of the Printer object is used to specify a printer driver. In early versions of Windows, and in MS-DOS, printer drivers were necessary to translate output in a way that each specific brand and model of printer could understand. By the time of Visual Basic 6.0, this was largely unnecessary, but the property was maintained for backward compatibility.

In Visual Basic 2005, the DriverName property no longer exists; printer drivers are managed by Windows, and you can no longer specify different drivers.

*hDC property*

In Visual Basic 6.0, the hDC property of the Printer object specifies a handle to a device context (a link between a Windows-based application, a device driver, and an output device, such as a printer).

In Visual Basic 2005, the hDC property no longer exists; an instance of a PrintDocument component is the equivalent of a device context.

*Page Property*

In Visual Basic 6.0, the Page property returns a count of pages that have been printed since your application started or since the last time the EndDoc statement was used on the Printer object. This property is often used to add a page number to each page during printing.

In Visual Basic 2005, page numbers are not tracked; however, you can easily keep a count by setting a variable in the BeginPrint event and incrementing it in the PrintPage event.

*Port Property*

In Visual Basic 6.0, the Port property returns the name of the port through which a document is sent to a printer.

In Visual Basic 2005, the Port property no longer exists; the PrintDialog and PrintPreviewDialog controls automatically manage port information.

*RightToLeft Property*

In Visual Basic 6.0, the RightToLeft property determines how the Printer object will format output on a bi-directional platform, such as Arabic Windows 95 or Hebrew Windows 95.

In Visual Basic 2005, the RightToLeft property is no longer necessary; the direction of printing is controlled by the localization settings in later versions of Windows.

*TrackDefault Property*

In Visual Basic 6.0, the TrackDefault property determines whether a Printer object always points to the same printer, or whether it changes the printer it points to if you change the default printer setting

in the operating system's Control Panel. Changing the TrackDefault property setting while a print job is in progress causes printing to halt immediately.

In Visual Basic 2005, the TrackDefault property no longer exists; the IsDefaultPrinter property of the PrinterSettings class can be used to determine if a printer is the default, but printing is no longer halted if the default printer changes.

*Zoom Property*

In Visual Basic 6.0, the **Zoom** property determines the percentage by which printed output is to be scaled up or down. For example, consider a letter-sized page printed with **Zoom** set to 50. This page contains as much data as a page of the size 17 by 22 inches because the printed text and graphics on the letter-sized page are scaled to one-half their original height and width.

In Visual Basic 2005, the **Zoom** property no longer exists; if a printer has zoom capabilities, settings are automatically exposed in the **Print** dialog box. You can also use graphics methods to scale the output prior to assigning it to a **PrintDocument** component.

*Graphics Properties and Methods*

In Visual Basic 6.0, various graphics properties and methods can be used to draw lines, shapes, and text on a **Printer** object.

In Visual Basic 2005, most objects no longer have their own graphics properties or methods; you can still draw lines, shapes, and text by creating and using a Graphics object. For more information, see Graphics for Visual Basic 6.0 Users.

**Check Your Progress**

Fill in the blanks:

1. Visual basic project basically consists of three parts - the user interface, the processing of information, and the .....
2. Own ..... can be created for the Visual basic IDE.
3. The Visual Basic 6.0 Printer object is replaced by the ..... component in Visual Basic 2005.

## 3.4 LET US SUM UP

This lesson is the soul of visual Basic Programming. In Start where we see the concept of forms and how we deal a project with multiple forms. Since any project or website contain more than one form so, it was important to remember the basic things when you deal with multiple from. After this we see the concept of visual basic Object Concept by which we able to understand the programming procedure, and the idea use to solve any problem/project. Later we see the concept of collection and how we use collect in our project and what is the advantage of using collection.

The heart and soul of Visual basic means concept of OOPS. In which we learn the basic concept of OOPS. This shows the idea that how we have to think to solve any problem using OOPS concept. After this we see the fundamental of class. We can say that in this topic we really understand the OOPS procedure and its advantage.

## 3.5 KEYWORDS

*Form Module:* It contains the graphic elements of the VB application along with the instruction.

*General Module:* It contains general-purpose instructions not pertaining to anything graphic on-screen.

*Class Module:* It contains the defining characteristics of a class, including its properties and methods.

*Global Module:* It contains declaration and procedures.

*Resource Files:* It allows you to collect all of the texts and bitmaps for an application in one place.

*Standard EXE:* It is a typical Application.

## 3.6 QUESTIONS FOR DISCUSSION

1. What are the Modules which are available in Visual Basic project?
2. Discuss the project templates will be displayed when you open visual basics.
3. Explain the project using multiple forms.
3. What is printer object in Visual basics? Explain the difference between visual basics 2005 and visual basics 6.0.

**Check Your Progress: Model Answers**

1. storage of information
2. Add-Ins
3. PrintDocument

## 3.7 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# LESSON

# 4

# ADVANCED PROGRAMMING TECHNIQUES

## CONTENTS

# 4.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of arrays
- Discuss how to identify the development needs
- Describe the significance of build in functions
- Identify and explain the user defined functions and procedures

# 4.1 INTRODUCTION

Whenever you are entering data, creating files or databases, you are working with text strings. Text strings contain characters that can be copied, deleted, cut and reassembled but they also have important visual characteristics: size, color, weight, transparency, etc. In this lesson we will look at different ways of manipulating those text strings.

# 4.2 ARRAYS

Now we are going to understand the concept of array. We will learn the differences between a fixed-size and dynamic array, how to properly declare each one, how to access them, how to loop through them, how to erase them, and a few other things. This tutorial applies to all versions of Visual Basic, however, versions before Visual Basic 6.0 do not include the split and join function.

## 4.2.1 Types of Arrays

- Fixed-Size Arrays
- Dynamic Arrays
- Retrieving the Contents of an Array
- Adding New Elements on the Fly
- Erasing an Array
- The Split Function
- The Join Function
- Multi-dimensional Arrays

*Fixed-Size Arrays*

A fixed-size array most closely matches our CD rack anology. There are a limited number of slots you can slide CDs into. Pretend you have three CDs - one by the Deftones, another by Tool, and a third by Disturbed. To fit all of these in your rack, the rack must contain at least three slots. So you declare your CD rack as having three slots:

```
Dim strCDRack(0 to 2) As String
```

You've just made a variable 'strCDRack' that contains three slots (#0, #1, and #2) and is of a String data type. Now you can insert your CDs into it:

```
Dim strCDRack(0 to 2) As String
    strCDRack(0) = "Deftones"
    strCDRack(1) = "Tool"
    strCDRack(2) = "Disturbed"
```

Notice that each of the three new lines starts off with the variable name and then gives an element number before having a value assigned. This is like numbering the slots on your CD rack starting at 0 up to 2 and then inserting a CD into each slot.

The format for declaring an array is:

Dim|Public|Private ArrayName(Subscript) As DataType

- Dim, Public, and Private declare the array and its scope. Using Dim in a procedure will make the array only available from within that procedure. Using it in the General Declarations section will make it available to all procedures in that module. Private has the same effect and should be used only at the modular level. Using Public will make the array available throughout the project.
- ArrayName is the name of the array.
- Subscript is the dimensions of the array.
- DataType is any valid data type.

*Dynamic Arrays*

The new Charlotte Church CD came out but your rack only has three slots. You don't want to throw away any of your CDs to make room for the new one so you decide to use your ultimate building skills to attach another slot. You start building:

```
Dim strCDRack() As String
ReDim strCDRack(0 to 2) As String
strCDRack(0) = "Deftones"
strCDRack(1) = "Tool"
strCDRack(2) = "Disturbed"
```

What have you done? Nothing wrong, you've just dimensioned your array another way that allows for expansion. Notice that the subscript of the Dim statement is missing. This is OK; it tells VB that your array is a dynamic array, meaning that you can change its size with ReDim.

Now that you've rebuilt the structure of your CD rack, allowing for expansion, it is time to expand:

```
Dim strCDRack() As String
ReDim strCDRack(0 to 2) As String
strCDRack(0) = "Deftones"
strCDRack(1) = "Tool"
```

```
strFriends(5) = "Carolyn"
strFriends(6) = "Kate"
```

For lngPosition = LBound(strFriends) To UBound(strFriends)

MsgBox strFriends(lngPosition)

Next lngPositionlngPositionlngPosition

There are two new functions in that snippet of code. LBound and UBound are used to dete ıine the lower and upper bounds of an array. Because strFriends has a lower bound of 0 and an upper bound of 6. These functions allow you to iterate through an array with a dynamic size and they keep you from having to keep track of the array's size yourself. With each iteration of that loop, lngPosition will count up from 0 to 6. By accessing the array as strFriends(lngPosition) you are greatly reducing the amount of code you have to write.

### *Adding New Elements on the Fly*

Sometimes you have an array that needs to keep growing, and you don't know what the upper bound will end up being. Maybe you are making a crappy MP3 player and need to ask the user to input song names. You might do something like this:

```
Dim strSongNames() As String  'Array of song names
Dim blDimensioned As Boolean 'Is the array dimensioned?
Dim strText As String 'To temporarily hold names
Dim lngPosition as Long 'Counting
'The array has not yet been dimensioned:
blDimensioned = False
Do
'Ask for a song name
strText = InputBox("Enter a song name:")
If strText <> "" Then
'Has the array been dimensioned?
If blDimensioned = True Then
'Yes, so extend the array one element large than its current upper bound.
'Without the "Preserve" keyword below, the previous elements in our array
would be erased with the resizing
    ReDim Preserve strSongNames(0 To UBound(strSongNames) + 1) As String
      Else
      'No, so dimension it and flag it as dimensioned.
      ReDim strSongNames(0 To 0) As String
      blDimensioned = True
End If
'Add the song name to the last element in the array.
```

```
strSongNames(UBound(strSongNames)) = strText
End If
Loop Until strText = ""
'Display entered song names:
For lngPosition = LBound(strSongNames) To UBound(strSongNames)
MsgBox strSongNames(lngPosition)
Next lngPosition
'Erase array
Erase strSongName
```

Look to the comments for an explanation of what is going on.

*Erasing an Array*

You should always erase your array when you are done using it, especially if you are using dynamic arrays. It's rather easy:

```
Dim strFriends(0 to 2) As String
strFriends(0) = "Bianca"
strFriends(1) = "Jeana"
strFriends(2) = "Erin"
Erase strFriends
```

*Split Function*

Sometimes we run into situations where we want to take the information from within a given string, separate it into multiple strings, and then place those strings in an array. For example, say we had this code:

```
Dim cdList As String
cdList = "Nevermind, OK Computer, I Care Because You Do, Icky Thump"
```

It'd be nice if we could easily take that list and put it in an array, wouldn't it? This could be done by using Visual Basic's built in string functions, however, writing and updating that code could prove to be time consuming and tedious. Luckily for us, Visual Basic 6.0 provides a built in function called split that allows us to easily parse out information from a string and place it into an array. It has the following syntax:

```
ArrayName = split(Sting Input[, Delimiter[, Length Limit[, Compare Mode]]])
```

String Input is the string that you want to parse.

Delimiter is an optional parameter that indicates what type of string separates the elements in the input string. By default this parameter is set to " ". That would mean an input string of "This is a test" would yield an array of 4 elements ("This", "is", "a", "test").

Length Limit is the maximum size your output array can be. The text remaining to be parsed will be set as the final element in the array.

Compare Mode. By default, Visual Basic compares strings character by character using their ASCII values. However, you can use different modes that will cause Visual Basic to compare strings differently. For example, vbTextCompare causes string comparisons to be case insensitive. This parameter effects how the Delimiter parses Input String.

The following is an example showing how to parse the list we showed earlier:

```
Dim strCDRack() As String
Dim cdList As String
Dim i As Integer
cdList = "Nevermind, OK Computer, I Care Because You Do, Icky Thump"
strCDRack = Split(cdList, ", ")
For i = LBound(strCDRack) To UBound(strCDRack)
MsgBox strCDRack(i)
Next
```

### *Join Function*

The split function allowed us to break strings down into arrays, is there a function that allows us to take arrays and make them one big long string? Yes, yes there is, and it is called join. join is a very simple function. It has the following syntax:

```
StringName = join(Array Input[, Delimiter])
```

Array Input is the array that you want to place into a string.

Delimiter is an optional parameter that indicates what you want to place between elements are added to the string. By default this parameter is set to "".

Using one of our previous examples, here is some sample code on how one might use join:

```
Dim strFriends(0 to 6) As String, lngPosition as Long
strFriends(0) = "Bianca"
strFriends(1) = "Jeana"
strFriends(2) = "Sam"
strFriends(3) = "Jenna"
strFriends(4) = "Erin"
strFriends(5) = "Carolyn"
strFriends(6) = "Kate"
Dim myFriends As String
'This will produce the following string: "Bianca, Jeana, Sam, Jenna, Erin,
Carolyn, Kate"
myFriends = Join(strFriends, ", ")
MsgBox myFriends
```

### *Multi-dimensional Arrays*

Now that you know what an array is, figuring out what a multi-dimensional array is will be a little easier. A multi-dimensional array is an array that looks like a table. If you have ever used Microsoft Excel, you would know. It can also be referred to an array of arrays. That is many arrays using the same name. Here is an example of a two-dimensional array.

Static iArray(1 To 2, 1 To 3)

All the elements list out are:

iArray(1,1), iArray(1,2), iArray(1,3), iArray(2,1), iArray(2,2), iArray(2,3)

Most of the time you will be using one dimensional arrays but some of the time it will be helpful to use 2 or even 3. More then 3 is not always a good idea because it tends to be hard to debug. The usefulness of VB will allow you to use as many as you want, each separated by commas. Keep in mind the more you use, the more computer memory you use up. This can get real dangerous real fast. If you have some free time someday go ahead and make and 14-15 dimensional array. Make sure you close out and save everything else before hand. See how well your computer runs. Be prepared to have to restart.

## 4.2.2 Passing Arrays as Parameters

1. Reference the array by name in the CALL statement
2. The called SUB/FUNCTION MUST accept the data into a parameter declared with the SAME data type, but without an index.
3. Arrays must be passed using the ByRef Method

```
Public Sub ProcTest(By Ref lngNum() as Long)
lngNum(1) = 2
End Sub
Public Sub Something()
 Dim lngArray(1 To 5) as Long
 Call ProcTest(lngArray)
 MsgBox("Index 1 = " & lngArray(1))
End Sub
```

## 4.2.3 Control Array

A group of data items that are referred to by the same name is known as array. whereas Control array is: A group of controls sharing the same name and event procedures

Index - A variable used to refer to each element in the control array.

Control Arrays can be used with the following controls:

- Text Boxes
- Option Buttons
- Check Boxes

- Labels
- Command Buttons

Control Arrays are most commonly used with text boxes and option buttons.

*Creating a Control Array*

Drop a control of the form and assign it a name

Drop a second control on the form and assign it the SAME name

You will be given a message saying that you already have a control named 'whatever' and asks you if you want to create a control array

Press "Yes" and this will create your control array.

Now, for each subsequent item added, give it the same name also.

Once you create the control array, each item within the array will have an index. The first will be an index of zero, 2nd of 1, 3rd of 2 and so forth...

***Example:*** Control Array using a Select Case

This example is if you had 6 option buttons with names of colors and a Rounded Square shape that will fill the color of the selected option button.

```
Private Sub optColors_Click(Index As Integer)<br>
    Select Case Index<br>
Case 0: shpRoundSquare.FillColor = vbBlack<br>
Case 1: shpRoundSquare.FillColor = vbBlue<br>
Case 2: shpRoundSquare.FillColor = vbYellow<br>
Case 3: shpRoundSquare.FillColor = vbGreen<br>
Case 4: shpRoundSquare.FillColor = vbRed<br>
Case 5: shpRoundSquare.FillColor = vbMagenta<br>
End Select<br>
End Sub
```

***Sum and Average of Test Scores Using Control Arrays***

Here is another example of a control array. This will be if you wanted to figure the sum and average of test scores.

```
'Calculating Sum and Average < br >
Private Sub cmdCalculate_Click()<br>
For X = 0 to 9 <br>
Sum = Sum + val(txtTest(X))<br>
Next X<br>
Average = Sum / 10<br>
End Sub
'Clearing text boxes and totals < br >
```

```
Private Sub cmdClear_Click()<br>
For Z = 0 to 9<br>
TxtTest(Z) = &quot;&quot;<br>
Next Z<br>
LblSum = &quot;&quot;<br>
LblAvg = &quot;&quot;<br>
Sum = 0<br>
Average = 0<br>
End Sub<br>.
```

## 4.3 POINTERS

A pointer is nothing more than a variable that holds the address in memory of another variable. In VB, pointers can only be used on value types and arrays. As a structure is a value type, pointers can be used with them, but there is one caveat with this, the structure must not contain any reference types if you plan to use pointers. Any of the following may be a pointer:

- Sbyte
- byte
- short
- ushort
- int
- uint
- long
- ulong
- char
- float
- double
- decimal
- bool

### *unsafe keyword*

Typically when we write code in VB by default it is what's known as safe code. Unsafe code allows you to manipulate memory directly, in the normal world of VB and the .NET Framework, this is seen as potentially dangerous, and as such you have to mark your code as unsafe. Using unsafe code, you are loosing garbage collection and whilst directly accessing memory, you have the possibility of accessing memory that you didn't want to and causing untold problems with your application. Unsafe code can only be used within a fully trusted assembly.

Now it is time to write some program code to demonstrate the use of pointers. Consider the following code.

```
Private Shared Sub Main(ByVal args As String())
    Dim age As Integer = 32
    Console.WriteLine("age = {0}", age)
End Sub
```

A very simple piece of code that will print age = 32 on the console. Now let's introduce a pointer into the mix. When we declare a pointer, we have to declare it in a certain way, this being type* variable; the asterisk (dereferencer symbol) informs the compiler that we are declaring a pointer variable, the type says that we intend to use a pointer to store the address of type type. So using our simple piece of code above, let's creates an integer pointer to point to age.

```
Private Shared Sub Main(ByVal args As String())
    Dim age As Integer = 32
    Dim age_ptr As Integer*
    Console.WriteLine("age = {0}", age)
End Sub
```

Notice the use of the unsafe keyword, if we had not enclosed our code within unsafe, we would have received the following compilation error.

Pointers and fixed sized buffers may only be used in an unsafe context.

We could have declared the whole method as unsafe and this would have worked fine, for example:

unsafe static Sub Main(ByVal args As String())

So, what can we do with our pointer? As stated at the beginning of the article, pointers hold addresses in memory to other variables, so we need to make age_ptr point to the memory location of age. To do this, we use the following line of code:

age_ptr = &age;

The ampersand '&' is the referencer and means the 'location of'. So if we now do the following in our code:

```
Private Shared Sub Main(ByVal args As String())
    Dim age As Integer = 32
    Dim age_ptr As Integer*
    age_ptr = AddressOf age
    Console.WriteLine("age = {0}", age)
    Console.WriteLine("age_ptr = {0}", *)
End Sub
```

We should now see the following output on the console:

Age = 32
age_ptr = 32

Basically, with the display line for age_ptr, we are printing the value held in the memory location that age_ptr is pointing to (in this instance, the age variable).

Now, what happens if we add the following to the example?

*age_ptr += 3;

Console.WriteLine("age now = {0}", age);

What do you think the output will be (note that we are printing the value of age)? If you have said 35 you are correct! The value of age has been changed to 35. Let's dissect the line *age_ptr += 3 and see what's happening. age_ptr has been declared as an integer, the term *age_ptr can be read as 'the integer value at the memory location age_ptr is pointing to'. We are then simply adding 3 to the integer value, giving us a result of 35. As age_ptr is pointing to the memory location where age stores its value, essentially we have modified the variable age.

### *Pointers to Structures*

As previously mentioned, you can define a pointer to a structure, but you have to make sure that there are no reference types in the structure, for example, consider the example code below:

```
Class TestClass
    ..........
End Class
Structure TestStruct
    Public age As Integer
    Public test As TestClass
End Structure
```

You would not be able to then do the following:

```
TStruct test_struct;
TStruct* struct_ptr;
struct_ptr = &test_struct;
```

When compiling this code, you would receive the error message:

Cannot take the address of, get the size of, or declare a pointer to a managed type.

So let's create a valid structure and see how we use a pointer to the structure. Our structure will be a simple one that contains two integers X and Y to simulate co-ordinates on a screen.

```
Structure Location
    Public X As Integer
    Public Y As Integer
End Structure
```

Now we can create the following code:

```
Private Shared Sub Main(ByVal args As String())
    Dim loc As Location
    loc.X = 100
```

```
    loc.Y = 100
     Dim loc_ptr As Location*
    loc_ptr = AddressOf
End Sub
```

There should be no surprises in the code above. loc_ptr is of the type Location, which is our structure. So how do we use our pointer? To access the X and Y members of our structure through the use of our pointer, we can use the following syntax

```
(*loc_ptr).X;  or  loc_ptr->X;
```

I know which one I would rather use (the second option), but both perform exactly the same function. So if we wanted to display and modify X we can do this as follows:

```
Private Shared Sub Main(ByVal args As String())
    Dim loc As Location
    loc.X = 100
    loc.Y = 100
    Dim loc_ptr As Location*
    loc_ptr = AddressOf
    Console.WriteLine("X = {0}", )
    loc_ptr->X = 200
    Console.WriteLine("X = {0}", )
End Sub
```

### *Pointers and Classes*

A class is a reference type, so therefore you cannot define a pointer to a class, but you can define a pointer to a class member that is a value type. Before we see an example of how to do this, there is something that you need to understand before creating pointers to value type members of a class.

When using safe code, the garbage collector will possibly move an object in memory during its lifetime, this is done to manage free resources and stop fragmentation. If you have a pointer to a value type in a class, this could cause some unexpected results, and some headaches! Needless to say, there is a way around this problem. We can instruct the garbage collector not to move specific objects. We do this by using the fixed keyword. Let's take a look at an example.

The design of the following class is not ideal and any OO design guru's out there will probably be shaking their heads at it, but it has been done as a very simple example.

***For Example***

```
Class PTest
    Public Sub New()
    End Sub
    Public age As Integer
```

```
    Public Sub DisplayAge()
        Console.WriteLine("age = {0}", age)
    End Sub
End Class
```

And some example code to create an instance of the class and then manipulate the public variable age using a pointer.

```
Private Shared Sub Main(ByVal args As String())
    Dim p As New PTest()
    p.age = 32
    p.DisplayAge()
    fixed (int* age = &p.age)
    *age += 3
    p.DisplayAge()
End Sub
```

Notice the use of the keyword fixed. Basically what this is ensuring is that during the execution of the code within the fixed statement, the object p will not be moved, thus solving the problem of our pointer pointing to an invalid memory location, this operation is called 'pinning'.

## 4.4 BUILT-IN-FUNCTIONS

These are the Built in Functions provided by Visual Basic. VB offers a rich set of built-in-functions for manipulating strings, numbers, dates and time. Built in functions are important and useful as they cut down effort and time, if one has to write the entire program all over. This topic is going to cover some of the important and commonly used string functions available in the Visual Basic Library.

### 4.4.1 String Functions

VB has numerous built-in string functions for processing strings. Most VB string-handling functions return a string, although some return a number (such as the Len function, which returns the length of a string and functions like Instr and InstrRev, which return a character position within the string). The functions that return strings can be coded with or without the dollar sign ($) at the end, although it is more efficient to use the version with the dollar sign.

*Len*

| Function: | **Len** |
|---|---|
| Description: | Returns a Long containing the length of the specified string |
| Syntax: | Len(string)<br>Where string is the string whose length (number of characters) is to be returned. |
| Example: | 1. lngLen = Len("Visual Basic") lngLen = 12 |

### *Mid$ (or Mid)*

| Function: | **Mid$ (or Mid)** |
|---|---|
| Description: | Returns a substring containing a specified number of characters from a string. |
| Syntax: | Mid$(string, start[, length])<br>The Mid$ function syntax has these parts:<br>string Required. String expression from which characters are returned.<br>start Required; Long. Character position in string at which the part to be taken begins. If start is greater than the number of characters in string, Mid returns a zero-length string ("").<br>length Optional; Long. Number of characters to return. If omitted or if there are fewer than length characters in the text (including the character at start), all characters from the start position to the end of the string are returned. |
| Example: | strSubstr = Mid$("Visual Basic", 3, 4) ' strSubstr = "sual"<br>Note: Mid$ can also be used on the left side of an assignment statement, where you can replace a substring within a string.<br>strTest = "Visual Basic "Mid$(strTest, 3, 4) = "xxxx"<br>'strTest now contains "Vixxxx Basic"<br>In VB6, the Replace$ function was introduced, which can also be used to replace characters within a string. |

### *Left$ (or Left)*

| Function: | **Left$ (or Left)** |
|---|---|
| Description: | Returns a substring containing a specified number of characters from the beginning (left side) of a string. |
| Syntax: | Left$(string, length)<br>The Left$ function syntax has these parts:<br>string Required. String expression from which the leftmost characters are returned.<br>length Required; Long. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in string, the entire string is returned. |
| Example: | 1. strSubstr = Left$("Visual Basic", 3)<br>2. 'strSubstr ="Vis"<br>3. 'Note that the same thing could be accomplished with Mid$:<br>4. strSubstr = Mid$("Visual Basic", 1, 3) |

### *Right$ (or Right)*

| Function: | **Right$ (or Right)** |
|---|---|
| Description: | Returns a substring containing a specified number of characters from the end (right side) of a string. |
| Syntax: | *Right$(string, length)*<br>The Right$ function syntax has these parts:<br>*string* Required. String expression from which the rightmost characters are returned. |

*Contd...*

| | |
|---|---|
| | *length* Required; Long. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in string, the entire string is returned. |
| Example: | 1. strSubstr = Right$("Visual Basic", 3) ' strSubstr = "sic"<br>2.<br>3.' Note that the same thing could be accomplished with Mid$:<br>4. strSubstr = Mid$("Visual Basic", 10, 3) |

## *UCase$ (or UCase)*

| Function: | **UCase$ (or UCase)** |
|---|---|
| Description: | Converts all lowercase letters in a string to uppercase. Any existing uppercase letters and non-alpha characters remain unchanged. |
| Syntax: | UCase$(string) |
| Example: | 1. StrNew = UCase$("Visual Basic") ' strNew = "VISUAL BASIC" |

## *LCase$ (or LCase)*

| Function: | **LCase$ (or LCase)** |
|---|---|
| Description: | Converts all uppercase letters in a string to lowercase. Any existing lowercase letters and non-alpha characters remain unchanged. |
| Syntax: | LCase$(string) |
| Example: | strNew = LCase$("Visual Basic") ' strNew = "visual basic" |

## *Instr*

| Function: | **Instr** |
|---|---|
| Description: | Returns a long specifying the position of one string within another. The search starts either at the first character position or at the position specified by the start argument, and proceeds forward toward the end of the string (stopping when either string2 is found or when the end of the string1 is reached). |
| Syntax: | InStr([start,] string1, string2 [, compare])<br>The InStr function syntax has these parts:<br>start Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. The start argument is required if compare is specified.<br>String 1 Required. String expression being searched.<br>String 2 Required. String expression sought.<br>compare Optional; numeric. A value of 0 (the default) specifies a binary (case-sensitive) search. A value of 1 specifies a textual (case-insensitive) search. |
| Examples: | 1. lngPos = Instr("Visual Basic", "a")<br>2. lngPos = 5<br>3.<br>4. lngPos = Instr(6, "Visual Basic", "a")<br>5. ' lngPos = 9 (starting at position 6)<br>6. |

*Contd...*

| | |
|---|---|
| | 7. lngPos = Instr("Visual Basic", "A")<br>8. ' lngPos = 0 (case-sensitive search)<br>9.<br>10. lngPos = Instr(1, "Visual Basic", "A", 1)<br>11. ' lngPos = 5 (case-insensitive search) |

### *InstrRev*

| Function: | **InstrRev** |
|---|---|
| Description: | Returns a Long specifying the position of one string within another. The search starts either at the last character position or at the position specified by the start argument, and proceeds backward toward the beginning of the string (stopping when either string2 is found or when the beginning of the string1 is reached).<br>Introduced in VB 6. |
| Syntax: | *InStrRev(string1, string2[, start, [, compare]])*<br>The InStr function syntax has these parts:<br>*String 1* Required. String expression being searched.<br>*String 2* Required. String expression sought.<br>*start* Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the last character position.<br>*compare* Optional; numeric. A value of 0 (the default) specifies a binary (case-sensitive) search. A value of 1 specifies a textual (case-insensitive) search. |
| Examples: | 'lngPos = InstrRev("Visual Basic", "a")'<br>'lngPos = 9<br>'lngPos = InstrRev("Visual Basic", "a", 6)'<br>'lngPos = 5 (starting at position 6)<br>'lngPos = InstrRev("Visual Basic", "A")'<br>'lngPos = 0 (case-sensitive search)<br>'lngPos = InstrRev("Visual Basic", "A",1)'<br>'lngPos = 9 (case-insensitive search)'<br>'Note that this last example leaves a placeholder for the start argument |

### *Notes on Instr and InstrRev:*

Something to watch out for is that while Instr and InstrRev both accomplish the same thing (except that InstrRev processes a string from last character to first, while Instr processes a string from first character to last), the arguments to these functions are specified in a different order. The Instr arguments are (start, string1, string2, compare) whereas the InstrRev arguments are (string1, string2, start, compare).

The Instr function has been around since the earlier days of BASIC, whereas InstrRev was not introduced until VB 6.

Built-in "vb" constants can be used for the compare argument:

vbBinaryCompare for 0 (case-sensitive search) vbTextCompare for 1 (case-insensitive search)

### *String $ (or String)*

| Function: | **String$ (or String)** |
|---|---|
| Description: | Returns a string containing a repeating character string of the length specified. |
| Syntax: | String$(number, character)<br>The String$ function syntax has these parts:<br>number Required; Long. Length of the returned string.<br>character Required; Variant. This argument can either be a number from 0 to 255 (representing the ASCII character code* of the character to be repeated) or a string expression whose first character is used to build the return string. |
| Examples: | strTest = String$(5, "a")'<br>strTest = "aaaaa"<br>strTest = String$(5, 97)'<br>strTest = "aaaaa" (97 is the ASCII code for "a") |

### *Space$ (or Space)*

| Function: | **Space$ (or Space)** |
|---|---|
| Description: | Returns a string containing the specified number of blank spaces. |
| Syntax: | Space$(number)<br>Where number is the number of blank spaces desired. |
| Examples: | 1. strTest = Space$(5) ' strTest = "  " |

### *Replace $ (or Replace)*

| Function: | **Replace$ (or Replace)** |
|---|---|
| Description: | Returns a string in which a specified substring has been replaced with another substring a specified number of times.<br>Introduced in VB 6. |
| Syntax: | Replace$(expression, find, replace with[, start[, count[, compare]]])<br>The Replace$ function syntax has these parts:<br>expression Required. String expression containing substring to replace.<br>find Required. Substring being searched for.<br>replace with Required. Replacement substring.<br>start Optional. Position within expression where substring search is to begin. If omitted, 1 is assumed.<br>count Optional. Number of substring substitutions to perform. If omitted, the default value is â€“1, which means make all possible substitutions.<br>compare Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. (0 = case sensitive, 1 = case-insensitive)Built-in "vb" constants can be used for the compare argument:<br>vbBinaryCompare for 0 (case-sensitive search)vbTextCompare for 1 (case-insensitive search) |

*Contd...*

| | |
|---|---|
| Examples: | strNewDate = Replace$("08/31/2001", "/", "-")<br>' strNewDate = "08-31-2001" |

### *StrReverse$ (or StrReverse)*

| | |
|---|---|
| Function: | **StrReverse$ (or StrReverse)** |
| Description: | Returns a string in which the character order of a specified string is reversed.Introduced in V' |
| Syntax: | StrReverse$(string) |
| Examples: | strTest = StrReverse$("Visual Basic") ' strTest = "cisaBlausiV" |

### *LTrim$ (or LTrim)*

| | |
|---|---|
| Function: | ***LTrim$ (or LTrim)*** |
| Description: | Removes leading blank spaces from a string. |
| Syntax: | LTrim$(string) |
| Examples: | strTest = LTrim$(" Visual Basic ") ' strTest = "Visual Basic " |

### *RTrim$ (or RTrim)*

| | |
|---|---|
| Function: | **Rtrim$ (or RTrim)** |
| Description: | Removes trailing blank spaces from a string. |
| Syntax: | Rtrim$(string) |
| Examples: | StrTest = RTrim$("Visual Basic") 'strTest = "Visual Basic" |

### *Trim$ (or Trim)*

| | |
|---|---|
| Function: | **Trim$ (or Trim)** |
| Description: | Removes both leading and trailing blank spaces from a string. |
| Syntax: | Trim$(string) |
| Examples: | 1. strTest = Trim$(" Visual Basic ") ' strTest = "Visual Basic<br>2. ' Note: Trim$(x) accomplishes the same thing as LTrim$(RTrim$(x)) |

### *Asc*

| | |
|---|---|
| Function: | **Asc** |
| Description: | Returns an Integer representing the ASCII character code corresponding to the first letter in a string. |
| Syntax: | Asc(string) |
| Examples: | 1. intCode = Asc("*") ' intCode = 42<br>2. intCode = Asc("ABC") ' intCode = 65 |

*Chr$ (or Chr)*

| Function: | Chr$ (or Chr) |
|---|---|
| Description: | Returns a string containing the character associated with the specified character code. |
| Syntax: | Chr$(charcode)Where charcode is a number from 0 to 255 that identifies the character. |
| Examples: | 1. strChar = Chr$(65) ' strChar = "A" |

## 4.4.2 Nifty Replace Function

*Strcmp Function*

According to the definition, the strcmp returns a zero when the strings match. But the problem is that whenever the strings dont match, it returns only two values either 1 or -1 depending upon the 2 strings. Whereas, the definition says that when the strings dont matched, they will return the numeric difference between the Ascii value of the first character of the each string that dont match.

for e.g.:- say strcmp("Ferry", Merry"),according to definition it must

return --> ascii(F)- ascii(M)

Like

The Like operator is not particularly fast. Consider alternatives. We don't have a generic rule to follow here. You need to measure the performance differences between your alternatives. Here is one rule though. It applies if you're looking for a certain string inside another one.

Instead of:

If Text$ Like "*abc*" Then

Use:

If InStr(Text$, "abc") < > 0 Then

## 4.4.3 Numeric Functions

VB supports many mathematical or numeric functions that can make calculations very simple, as you just have to feed in the variables, and get the output after the function processes it.

*Val Function*

This function is used as a converter function. It converts the numbers contained in a string to its numeric equivalent. it just is the opposite of the Str() function, however it has some differences too.

Syntax:

.....

```
Val(String)
```

*Note:* the string must be put in quotes

Working:

if we feed it a value like this

Val("124566")

will produce the number 124566and,

Val("125.0066")

will produce the number 125.0066

However, the Val function stops reading the string, when it encounters a character which has NO NUMERIC EQUIVALENCE, which is for alphabets, symbols, commas, etc are not recog ed. But, Blanks, Tabs, and linefeed characters are simply removed and not counted . included at all.

So therefore we get something like this

Val ("4566 3442 5553")

gives the output 456634425553, no tabs, spaces included.

Now, for input something like this it has another output

Val ("12534ABCD2345")

gives output 12534. Why? Because when it reads "A" after four, it simply doesn't recognize it as a number, and the string is not read further, so even if there are numbers at the back of the string, they aren't added because they are never read.

Now for more information, this function also recognizes the radix prefixes &O for (Octal numbers) and &H for (Hexadecimal numbers) for example

Val("&H2D")

returns its decimal equivalent, which is 45.

The main use of Val function is to obtain numeric values from string variables, such as values inputted/obtained from a text box.

CODE

```
Dim N1, N2, N3 as Integer
N1 = Val(text1.text)
N2 = Val(text2.text)
N3 = N1*N2
text3.text = Str(N3)
```

Now if you remove the "Val" functions, you will get a "Type mismatch" error in runtime. however you will get the same error if you do not convert it back to string when passing it to a string variable, but if you are directly passing it a textbox, then the Str() function isn't really necessary.

***Sgn Function***

This function is used to determine the sign of a number.

Syntax:

```
Sgn(number)
```

*Note:* the "number" should be a valid number or numeric expression

now there are three cases for a sign of a number

If the number is

Positive (number > 0) then sign value returned is "1"

Negative (number < 0) then sign value returned is "-1"

Zero (number = 0) then sign value returned is "0"

*Example:*

```
Dim N1, N2, N3 As Integer
N1 = 234
N2 = -19
N3 = 0
Print Sgn(N1)
Print Sgn(N2)
Print Sgn(N3)
the output is
1
-1
0
```

*Int and Fix Functions*

Both these functions are used to remove the numbers left to the decimal point. But these functions work differently.

the Fix() function simply removes (truncates) the fractional part.

num1 = Fix(134.2423)

'// num1 is stored with the value 134

and the Int() function removes the fractional part, and rounds down to the nearest integer value

num1 = Int(134.2423)

'// num1 is stored with the value 134

both Fix() and Int() functions behave the same for positive fractional numbers, but in case of negative input number, they differ

num1 = Int(-2554.23)

'// num1 is stored with the value -2554

But the Int() function:

num1 = Int(-2554.23)

'// num1 is stored with the value -2555

why? because Int() function removes the fractional part, and rounds down to the nearest integer value as known, so it is doing what it is supposed to do, as "-2555" is smaller than "-2554".

***Rnd Function***

The Rnd() function is used to generate a random number. This function returns a value equal or greater than zero, but lesser than 1.

Syntax:

Rnd([number])

*Note:* Here [number] denotes that it is optional argument

However the output is quite unpredictable, but it can be somewhat known on the basis of what input has been given.

if [Number] argument is:

1. ***Less than Zero***
   - the Rnd() function generates the same number every time using [number] as an initial value used to generate pseudorandom numbers (called Seed)
2. ***Equal to Zero***
   - the Rnd() function generates the most recently generated number, which remains the same
3. ***Greater than Zero***
   - the Rnd() function generates the next random number in the sequence.
4. ***Argument Not Supplied***
   - the Rnd() function generates the next random number in the sequence.

   Now we can modify the use in several ways to generate random numbers within a limit we can write it as

***Syntax:***

```
(( upperlimit - lowerlimit ) * Rnd + lowerlimit)
```

where upperlimit and lowerlimit are the range numbers in which random numbers have to be generated.

If you want integer numbers, simply put the Int() function in front of the expression

Int (( upperlimit - lowerlimit ) * Rnd + lowerlimit)

note that numbers generated WILL be BETWEEN the limits, as the limit numbers are excluded.

*Note:*

Before calling the Rnd() function, one should use the Randomize() statement without an argument to initialize the random number generator based on the system timer. But if you use the Randomize (number), then the Randomize Statement initializes the random number generator using the optional (number) argument as the seed value.

*Some Examples:*

```
Dim R_num as Integer
R_num = ((10 - 4)*Rnd +4)
Print R_num
R_num = Int((10-4)*Rnd + 4)
Print R_num
```

the output will be (one of the output possibilities)

9.171213 ⇓ generated due to the Rnd() function with limits

8 ⇓ same as before, but Int() function removes the decimal parts.

Also,

To repeat sequences of random numbers, call Rnd() with a negative argument immediately before using Randomize with a numeric argument. Using randomize with the same value for number does not repeat the previous sequence.

*Format Function*

Format function rearranges number, strings, dates or any other value in a "formatted" way. Mainly used to represent dates in different ways when needed.

Syntax:

```
Format(expression,[Format],[firstdayofweek],[firstweekofyear])
```

where,

expression - Required. this is the value which is to be formatted

[format] - Optional. A valid names or user defined format expression.

[firstdayofweek] - Optional. A constant that specifies the first day of the week

[firstweekofyear] - Optional. A constant that specifies the first week of the year.

The expression argument a number to convert and he format argument is a string made up of symbols that shows how to format the number. The most commonly used symbols are listed:

| Symbol | | Use |
|---|---|---|
| "0" | : | Digit placeholder; prints trailing or leading zeros into positions. |
| "#" | : | Digit placeholder; no trailing or leading zeros. |
| "." | : | Decimal placeholder. |
| (") | : | the " is a thousands placeholder/ |
| + $ () space | : | literal character characters are displayed exactly as typed into the format string |

These format options enable a user to display numbers in different ways when needed; one can specify the number of decimal places, leading or trailing zeros, currency formats and others. [/CODE]

*Number Formatting*

The following block will give information on how to format a number in several ways:

Format: (10203.192, "0000000.00000")

Result: 0010203.19200 // leading and trailing zeros

Format: (10203.192, "#####.#####")

Result: 10203.192 // no leading or trailing zeros

Format: (10203.192, "#####.00000")

Result: 10203.19200 // only trailing zeros

Format: (10203.192, "##,###.00000")

Result: 10,203.19200 // thousands comma inserted in number

Format: (10203.192, "$##,###.000")

Result: $10,203.192 // currency formatting

You can try as many combinations as possible, depending on what format you need.

The symbol for decimal separator is a period (.) and the thousands separator is the comma (,)

Also

*Note:* The conversions above are possible when the country in the Windows Control Panel is set to [English - (United States)].

Also the separator character depends on the selection of the country, for example many countries will use the imperial system of numbers, and other will use the metric system. In these cases the commas will be inserted according to the country selected in Windows.

### 4.4.4 Date and Time Functions

VB keywords that reference the current date and/or time:

*Now* Returns the current date and time together

*Date* Returns the current date

*Time* Returns the current time

For the examples that follow, assume that the current Date/Time (Now) is Friday, August 31, 2001 at 9:15:20 PM

The following functions isolate the date portion and time portion, respectively, of a Date/Time value.

Now Look, the following Function

*DateValue*

Returns the date portion of a Date/Time value, with the time portion "zeroed out". (Note: When the time portion of a date/time variable is "zeroed out", the time would be interpreted as 12:00 AM.)

*Example:*

```
Dim dtmTest As Date
dtmTest = DateValue(Now)
```

At this point, the date portion of dtmTest is 8/31/2001, with a time portion of 0 (12:00 AM midnight).

*TimeValue:* Returns the time portion of a Date/Time value, with the date portion "zeroed out". (Note: When a date/time variable is "zeroed out", the date will actually be interpreted as December 30, 1899.)

*Example:*

```
Dim dtmTest As Date
dtmTest = TimeValue(Now)
```

At this point, the time portion of dtmTest is 9:15:20 PM, with a date portion of 0 (12/30/1899).The following functions are used to isolate a particular part of a date:

*Weekday:* Returns a number from 1 to 7 indicating the day of the week for a given date, where 1 is Sunday and 7 is Saturday

*Example:*

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

*Note:* When necessary to refer to a day of the week in code, VB has a set of built-in constants that can be used instead of the hard-coded values 1 thru 7

Constant Value

```
vbSunday 1
vbMonday 2
vbTuesday 3
vbWednesday 4
vbThursday 5
vbFriday 6
vbSaturday 7
```

*WeekdayName:* Returns a string containing the weekday name ("Sunday" thru "Saturday"), given a numeric argument with the value 1 through 7.

*Example:*

```
strDOW = WeekdayName(6) ' strDOW = "Friday"
```

The WeekdayName function takes an optional, second argument (Boolean) indicating whether or not to abbreviate the weekday name. By default, the second argument is False, meaning do not abbreviate and return the full name. If True, the first three letters of the weekday name will be returned:

*Example:*

```
strDOW = WeekdayName(6, True) ' strDOW = "Fri"
```

You can nest the Weekday function within the WeekdayName function to get the weekday name for a given date:

*Example:*

```
strDOW = WeekdayName(Weekday(Now)) ' strDOW = "Friday"
```

***Month:*** Returns a number from 1 to 12 indicating the month portion of a given date.

*Example:*

```
intMonth = Month(Now) ' intMonth = 8
```

***MonthName:*** Returns a string containing the month name ("January" thru "December"), given a numeric argument with the value 1 through 12.

*Example:*

```
strMoName = MonthName(8) ' strMoName = "August"
```

The MonthName function takes an optional, second argument (Boolean) indicating whether or not to abbreviate the month name. By default, the second argument is False, meaning do not abbreviate and return the full name. If True, the first three letters of the month name will be returned.

*Example:*

```
strMoName = MonthName(8, True) ' strMoName = "Aug"
```

You can nest the Month function within the MonthName function to get the month name for a given date:

*Example:*

```
strMoName = MonthName(Month(Now)) ' strMoName = "August"
```

***Day:*** Returns a number from 1 to 31 indicating the day portion of a given date.

*Example:*

```
intDay = Day(Now) ' intDay = 31
```

***Year*** Returns a number from 100 to 9999 indicating the year portion of a given date.

*Example:*

```
intYear = Year(Now) ' intYear = 2001
```

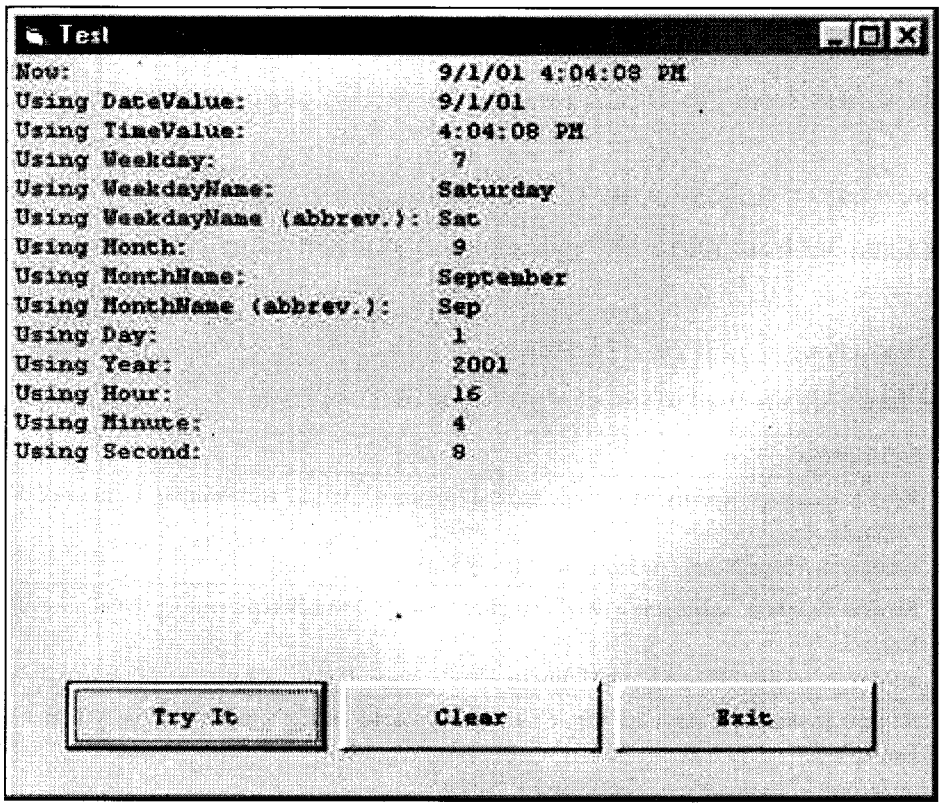
The following functions are used to isolate a particular part of a time:

| Function | Description |
|---|---|
| Hour | Returns an integer specifying a whole number between 0 and 23 representing the hour of the day.<br>Example:<br>intHour = Hour(Now) ' intHour = 21 (for 9 PM) |
| Minute | Returns an integer specifying a whole number between 0 and 59 representing the minute of the hour.<br>Example:<br>intMinute = Minute(Now) ' intMinute = 15 |
| Second | Returns an integer specifying a whole number between 0 and 59 representing the second of the minute.<br>Example:<br>intSecond = Second(Now) ' intSecond = 20 |

To demonstrate the date functions shown thus far, set up a "Try It" project, and place the following code in the cmdTryIt_Click event:

```
Private Sub cmdTryIt_Click()
Print "Now:"; Tab(30); Now
Print "Using DateValue:"; Tab(30); DateValue(Now)
Print "Using TimeValue:"; Tab(30); TimeValue(Now)
Print "Using Weekday:"; Tab(30); Weekday(Now)
Print "Using WeekdayName:"; Tab(30); WeekdayName(Weekday(Now))
Print "Using WeekdayName (abbrev.):"; Tab(30); WeekdayName(Weekday(Now),
True)
Print "Using Month:"; Tab(30); Month(Now)
Print "Using MonthName:"; Tab(30); MonthName(Month(Now))
Print "Using MonthName (abbrev.):"; Tab(30); MonthName(Month(Now), True)
Print "Using Day:"; Tab(30); Day(Now)
Print "Using Year:"; Tab(30); Year(Now)
Print "Using Hour:"; Tab(30); Hour(Now)
Print "Using Minute:"; Tab(30); Minute(Now)
Print "Using Second:"; Tab(30); Second(Now)
End Sub
```

OutPut Will be:



### 4.4.5 DatePart Function

The generic DatePart function returns an Integer containing the specified part of a given date/time value. Thus, it incorporates the functionality of the Weekday, Month, Day, Year, Hour, Minute, and Second functions. In addition, it can used to get the quarter of a given date (1 through 4), the "Julian" date (the day of the year from 1 to 366), and the week number (1 through 53).

Syntax:

```
DatePart(interval, date[,firstdayofweek[, firstweekofyear]])
```

The DatePart function syntax has these parts:

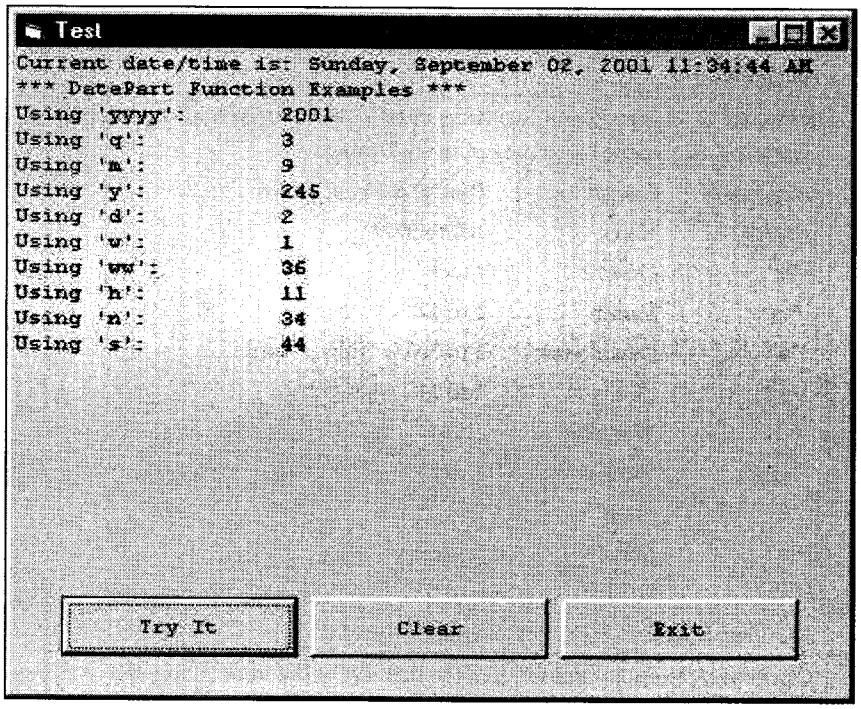
| Part | Description |
|---|---|
| *Interval* | Required. String expression that is the interval of time you want to return.<br>The string expression can be any of the following:<br>*Expression* — *Description* — *Possible Range of Values*<br>"yyyy" — Year — 100 to 9999<br>"q" — Quarter — 1 to 4<br>"m" — Month — 1 to 12<br>"y" — Day of year — 1 to 366 (a "Julian" date)<br>"d" — Day — 1 to 31<br>"w" — Weekday — 1 to 7<br>"ww" — Week — 1 to 53<br>"h" — Hour — 0 to 23<br>"n" — Minute — 0 to 59<br>"s" — Second — 0 to 59 |
| *Date* | Required. Date value that you want to evaluate. |
| *firstdayofweek* | Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed. |
| *firstweekofyear* | Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. |

To demonstrate DatePart, set up a "Try It" project, and place the following code in the cmdTryIt_Click event:

```
Private Sub cmdTryIt_Click()
Print "Current date/time is: "; _
Format$(Now, "Long Date"); _
Spc(1); _
Format$(Now, "Long Time")
Print "*** DatePart Function Examples ***"
Print "Using 'yyyy':"; Tab(20); DatePart("yyyy", Now)
Print "Using 'q':"; Tab(20); DatePart("q", Now)
Print "Using 'm':"; Tab(20); DatePart("m", Now)
Print "Using 'y':"; Tab(20); DatePart("y", Now)
Print "Using 'd':"; Tab(20); DatePart("d", Now)
```

```
Print "Using 'w':"; Tab(20); DatePart("w", Now)
Print "Using 'ww':"; Tab(20); DatePart("ww", Now)
Print "Using 'h':"; Tab(20); DatePart("h", Now)
Print "Using 'n':"; Tab(20); DatePart("n", Now)
Print "Using 's':"; Tab(20); DatePart("s", Now)
End Sub
```

Output will be:




*Piecing Separate Numbers together to Form a Date or Time Value*

In the previous examples, we saw ways to isolate parts of a date/time value. What if you need to go the "other way"? If you have the separate parts of a date/time value in different variables and want to piece them together to formulate a date or time, there are two functions you can use to do this: DateSerial and TimeSerial.

The DateSerial takes three numeric arguments: year, month, and day respectively. It returns a date based on those values.

*Example:*

```
Dim intYear As Integer
Dim intMonth As Integer
Dim intDay As Integer
Dim dtmNewDate As Date
intYear = 2001
intMonth = 9
```

```
intDay = 2
dtmNewDate = DateSerial(intYear, intMonth, intDay)
' returns 9/2/2001
```

The TimeSerial takes three numeric arguments: hour, minute, and second respectively. It returns a time based on those values.

*Example:*

```
Dim intHour As Integer
Dim intMinute As Integer
Dim intSecond As Integer
Dim dtmNewTime As Date
intHour = 11
intMinute = 34
intSecond = 44
dtmNewTime = TimeSerial(intHour, intMinute, intSecond)
'returns 11:34:44 (AM)
```

## 4.5 USER DEFINED FUNCTIONS & PROCEDURES

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

It is easier to debug a program a program with procedures, which breaks a program into discrete logical limits. Procedures used in one program can act as building blocks for other programs with slight modifications.

A Procedure can be Sub, Function or Property Procedure.

### 4.5.1 Sub Procedures

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

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

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

### 4.5.2 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.

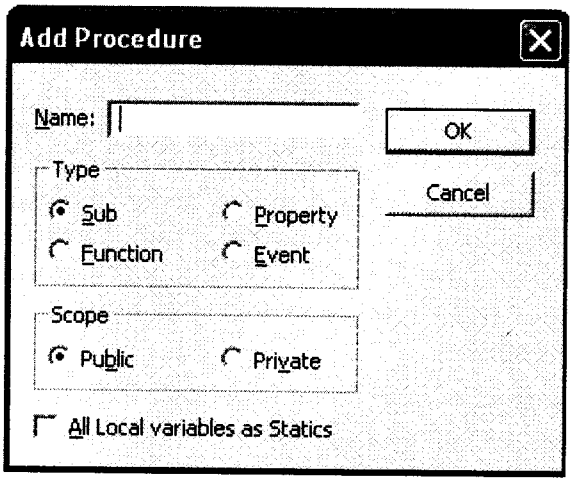
### 4.5.3 General Procedures

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

In order to add General procedure:

The Code window is opened for the module to which the procedure is to be added.

The Add Procedure option is chosen from the Tools menu, which opens an Add Procedure dialog box as shown in the figure given below.




The name of the procedure is typed in the Name textbox

Under Type, Sub is selected to create a Sub procedure, Function to create a Function procedure or Property to create a Property procedure.

Under Scope, Public is selected to create a procedure that can be invoked outside the module, or Private to create a procedure that can be invoked only from within the module.

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

### 4.5.4 Function Procedures

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

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

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

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

### 4.5.5 Property Procedures

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

### 4.5.6 Handling Function and Sub Procedures

First of all we will learn that how we pass a function and a procedure as a parameter.

*Passing Arguments*

This can be done in two ways: By Value, or By Reference.

- By Value means that a copy of the actual variable is passed. The variable itself will not change within the procedure or function.
- By Reference means that the memory-address itself is passed. The variable will change if it is altered within the procedure or function it's being passed to. By Reference is the VB standard.

Create a new standard Exe project. Now double click the form. This brings us into the Form_Load() sub procedure. We start by declaring two integer variables:

- Dim intValue as Integer
- Dim intReference as Integer

Next, we assign the two variables a value:

intValue = 10

intreference = 100

We'll create a new procedure now:

```
Private sub AddTen (ByVal int1 As integer, ByRef int2 As Integer)
int1 = int1 + int1
int2 = int2 + int2
```

To show the values in between, we're going to call in help from a debugging tool within VB the Immediate Window. We can use the Immediate window (among other things) to display values while an application is running. Type the following code within the procedure:

Debug.Print "Values of int1 in procedure: "; int1

Debug.Print "Values of int2 in procedure: "; int2

This will show us what the value of the two integers is after we multiplied the numbers within the procedure. Back to the Form_Load sub. Under the part where you set their values, insert the following code:

```
Call AddTen(intValue, intReference)
Debug.Print "Values of intValue after procedure:"; intValue
Debug.Print "Values of intReference after procedure:"; intReference
```

First we call the sub procedure, specifying the two parameters. intValue being the first one is passed by Value (See declaration of the function earlier. intValue is being passed as int1 to the procedure), while intReference is passed By Reference (int2)).

The second part is to show that the variables indeed changed. If you run the project now, the Immediate Window will show this:

Values of int1 in procedure: 20

Values of int2 in procedure: 200

Values of intValue after procedure: 10

Values of intReference after procedure: 200

Since intValue was passed by Value, the function altered a copy, and the actual variable itself remained the same. The intReference variable though changed not only within the sub procedure, but in the rest of the application as well.

Remember, if you want to pass variables to sub routines or functions, they become mandatory. Someone can't call the function and not specify parameters.

**Check Your Progress**

1. Define the work of UCase$ (or UCase) function.
2. Define the work of InstrRev Function.
3. State whether the following statements are true or false:
   (i) Now Returns the current date and time separate.
   (ii) Trim$(or Trim) Removes only leading blank space from a string.
4. Fill in the blanks:
   (i) The TimeSerial takes three numeric arguments: .....respectively.
   (ii) The ..... function generates the next random number in the sequence.

## 4.6 LET US SUM UP

User who want to build his career as developer for those this unit work as core of Visual basic language. Although if you know the evaluation and other basic things in visual basic but if your facts is week in this portion then you have to definitely strong this portion. This lesson holds the vital portion of visual basic. In the starting it depict all the string function with example after then it will show some good example date Function.

After then it cover another good topic, Function and Procedure. If you know the right time to use function then you will solve your half of the problem. Because, function provide a good concept of "reusability", which help you to reduce the length of the code.

In the last portion it focus on the array. Array is use to as data container, further you will study that in many place Array himself consider as Datastructure.

## 4.7 KEYWORDS

*Weekday: Returns a number from 1 to 7 indicating the day of the week for a given date.*

*DatePart:* It is a function which returns an Integer containing the specified part of a given date/time value.

*ARRAYS:* It is a list of same datatype.

*Enums (enumerations):* are structures you define when you need a set of constant values.

*List Box:* A type of control used to hold a list of items from which the user can select one item from the list.

## 4.8 QUESTIONS FOR DISCUSSION

1. Explain the work of the following function:
   (a) String$ (or String)
   (b) Strcmp function
   (c) Replace$ (or Replace)
2. Explain the working of numeric functions.
3. What is the difference between DateValue and DatePart Function?
4. What is the sub-procedure? Give an example.
5. What are the different kinds of array? Explain with example.

**Check Your Progress: Model Answers**

1. Converts all lowercase letters in a string to uppercase. Any existing uppercase letters and non-alpha characters remain unchanged.
2. Returns a Long specifying the position of one string within another. The search starts either at the first character position or at the position specified by the start argument, and proceeds forward toward the end of the string (stopping when either string2 is found or when the end of the string1 is reached).
3. (i) False
   (ii) False
4. (i) hour, minute, and second
   (ii) Rnd()

## 4.9 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# UNIT III

# LESSON

# 5

# OBJECTS

CONTENTS



## 5.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of objects
- Discuss the manipulation of objects in visual basics
- Describe the significance of collections
- Create an object in visual basics
- Discuss the various steps in building classes

## 5.1 INTRODUCTION

Generally, objects in any language supporting OOPs are the variables belonging to a particular user defined data type. These user defined data types are known as Classes. In Visual Basic, the controls on the toolbox represent individual classes and, are called as objects, when placed on a form.

Each time we select a control and place it on the form, an object to that class is formed. A CommandButton. TextBox, etc., and even the form itself are the instances of some class. These are known as objects as soon as they are included in the project.

Technically, a class is a way of binding data describing an entity and its associated function together. Classes are the media used to represent real world entities that not only have data type properties but also associated operations. An object consists of properties and methods, just like other Visual Basic elements.

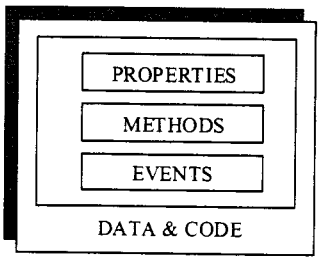



**Figure 5.1 Object**

Since objects have been called a piece of code, you can distinguish them from procedures if they have three characteristics: Inheritance, Encapsulation and Polymorphism. Visual Basic's objects follow inheritance and encapsulation but not polymorphism.

Every object used in Visual Basic has its own class name to distinguish it from other objects. Properties and methods constitute the object's interface. This interface let's you talk and command the object and access it. You can't access the object's code or data directly. For example, text stored in a text box cannot be fetched directly unless you use the Text property.

## 5.2 VB OBJECT

### 5.2.1 Object Browser

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

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

*Object naming Conversions of Controls (prefix)*

| | |
|---|---|
| Form | -frm |
| Label | -lbl |
| TextBox | -txt |
| CommandButton | -cmd |
| CheckBox | -chk |
| OptionButton | -opt |

| | |
|---|---|
| ComboBox | -cbo |
| ListBox | -lst |
| Frame | -fme |
| PictureBox | -pic |
| Image | -img |
| Shape | -shp |
| Line | -lin |
| HScrollBar | -hsb |
| VScrollBar | -vsb |

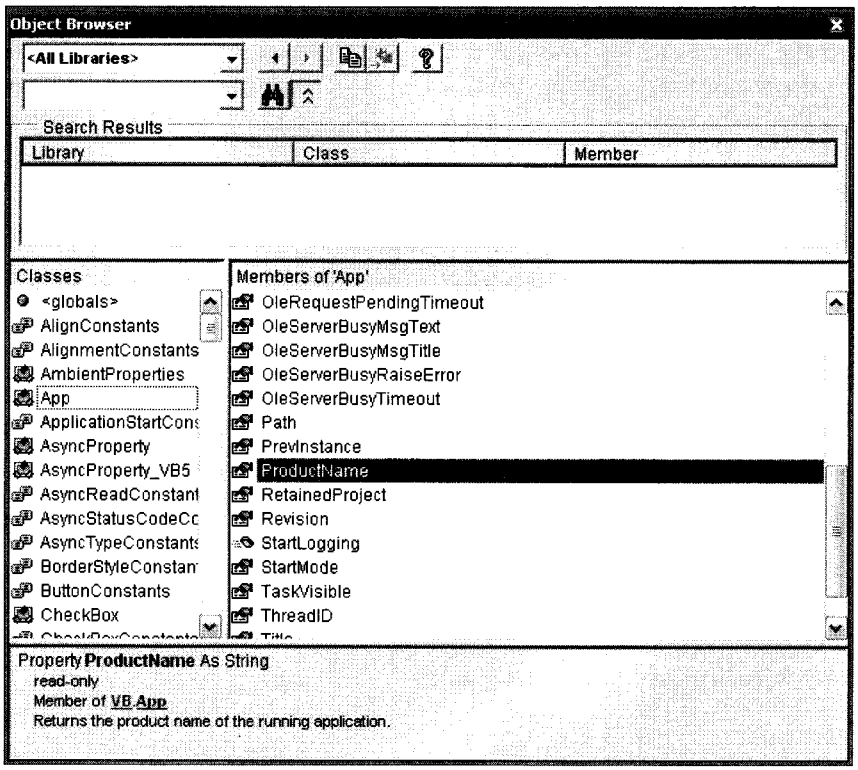
Visual Basic and Visual Basic for Applications (Microsoft Office VB Editor) comes with an Object Browser.

This includes any references or controls you have added to the project, as well as all the forms, modules and classes. The browser is a great way to navigate your objects without having to open each one individually.

Now we will learn how to explore all of the exposed members any objects in your project by using the Object Browser. It would probably be a good idea for you to open your Object Browser so you can explore at the same time.

### *Window*

The Object Browser window can be accessed by pressing F2 from within the IDE, or through the View menu.



Each of the various items in the list have icons representing their types.

The very top of the window has controls that enable you to filter the view. By default, the Object Browser shows items for all libraries in the project. This includes references (including the VB runtime), controls, forms, modules and classes. If you are using the Visual Basic Editor within a Microsoft Office application (VBA), it also shows such items from the application itself. The parent application in the VBA environment is always shown as the Application class.

We can also search, navigate backward and forward through ones you've already viewed, copy the item to memory, and view the Help topic on the selected item. If the currently selected item is, or is part of, a form, module or class in your project, the View Definition button will also be available.

*Using the Browser*

***Window Overview:*** In the screenshot above, the ProductName property of the App class is selected. At the bottom of the window, information is displayed for the selected item with the item name in bold. It shows what type of item it is (Property) and what its datatype is (String). There's also the indication "read-only" below the definition. This lets you know that you can only Read the property and not Set it. Example:

*Code*

'This is correct

```
strName = App.ProductName
```

'This is NOT correct and will generate an error

```
App.ProductName = "Bogus Application"
```

Below the "read-only" line, it lets you know that ProductName is a member of VB.App. App is the class as displayed in the list on the left and VB is the library. If you click on the VB link, it will filter the "Classes" list to show only those in the VB library. You can reset it by selecting <All Libraries> from the dropdown list at the top, or click the Back button.

*Quote*

Returns the product name of the running application.

This is a very handy feature. You can possibly find a member you want just by browsing through and reading the descriptions.

*Members with Parameters*

If a member has parameters, they will be shown in the same line as the member definition. For example, if the LogEvent method of the App class is selected, it looks like this:

Sub LogEvent(LogBuffer As String, [EventType])

From this, we know that LogEvent is a Sub - which is called directly and no value is returned. Also, it expects a String value for the first parameter and optional (indicated by the brackets) EventType parameter. This declaration does not state what EventType is, so by pressing F1 to open help on LogEvent, we can see what the valid parameters are.

If the member is a Function, it will usually contain a datatype for the return value. If it's not an object, it will just state the datatype (ex: String, Long, etc). However, if it's an object/class type, the type will be displayed as a link. For example, by selecting Clipboard under Classes, then GetData under Members, we see:

*Quote*

Function GetData([Format]) As IPictureDisp

The IPictureDisp part is a link. If you click it, the browser will select IPictureDisp in the Classes list. You can then return back to the GetData function by clicking the Back button at the top of the browser. When the return value is a class such as this one, you have to use the Set statement. Examples:

*Code*

```
Dim objPic As IPictureDisp
Set objPic = Clipboard.GetData()
MsgBox objPic.Width & "x" & objPic.Height
'....or....
Set Picture1.Picture = Clipboard.GetData()
```

***Finding Related Members***

Many times a constant value is seen in sample code such as vbNewLine, vb3DFace or vbMagenta. Where do these come from? What other ones are available? The Visual Basic runtimes come with thousands (just a guess) of constant values such as these. Each of them are available under Enumerations or Modules. Since very few people include the enumeration or module name, it's hard to tell where these values come from and what other options there are. This is where the Object Browser search can be useful.

The search box is located directly under the Library dropdown list at the top of the browser window. Beside it is a button with a binoculars image. If you put vbNewLine into the search, then press Enter or click the button, the Search Results panel will display (if not already visible). It shows us that vbNewLine is a member of the VBA.Constants module. It also navigates to it in the list. Notice that there are other related constants in the module.

If you perform the search on vb3DFace, it will result in VBRUN.SystemColorConstants. Notice that a description is even provided for the constant. A similar result will appear for vbMagenta.

If you want to narrow your browsing, you can change the library being viewed. For example, if you have added "Microsoft Active Data Objects 2.8 Library" (msado15.dll) to your project, you can select ADODB from the Library dropdown to see all classes, enumerations, etc available in the library. This feature makes it real easy to determine what is available in a Control or Reference that you've never used before. By narrowing it to the library, you can see what all it has to offer. If the library or control was developed completely (correct datatypes, descriptions for all methods, etc), you will probably be able to figure out how to use it without any documentation.

**Object Browser Shortcut Keys, Visual Basic 6.0 Default Shortcut Option**

| Command | Shortcut Keys | Description |
|---|---|---|
| Edit.FindSymbol | CTRL + SHIFT + Y<br>CTRL + SHIFT+ ALT + F12 | Displays the Find Symbol dialog box |
| View.FindSymbolResults | CTRL + ALT + Y | Displays the Find Symbol Results window. |
| View.ObjectBrowserBack | ALT + - | Moves back to the previously selected object in the selection history of the Object Browser |
| View.ObjectBrowserForward | SHIFT + ALT + | Moves forward to the next object in the selection history of the Object Browser |

## 5.2.2 General Object Variables

Visual Basic is an (OO) object-oriented language. Performing a task in Visual Basic (VB) or Visual Basic for Applications (VBA) involves manipulating various types of objects, each of which may have several different properties and methods. To perform a task using VBA you return an object that represents the appropriate Excel element and then manipulate it using the objects, methods and properties.

*Objects*

A simple statement Range("A1"). Select illustrates an important characteristic of VB. The syntax of many statements first specifies an object, Range("A1"), and an action upon it, Select. An object is a special type of variable that contains both data and code and represents an element of Excel. Objects exist only in the computer's memory; they don't appear in your code. There are many types of objects in VB, and even more in Excel VBA. Excel objects that you will encounter include the Application object which represents the Excel application itself, the Worksheet object representing a worksheet, and the Range object representing an individual cell or a rectangular range of cells (e.g. cells A2:C9 of the currently active worksheet). Objects have types just as variables have data types. Object types are called classes. Workbook, Worksheet and Range are just a few of Excel's object classes.

*Methods*

In VB an action that an object can perform is referred to as a method. Thus a method of an object is a procedure that carries out some action involving the object. Consider the object Dog. To cause it to bark we could write Dog.Bark.

However a Dog is capable of more than barking, for example we could have Dog.Sit, Dog.Fetch. Like any other procedure, a method of an object can take one or more arguments, as in this example:

ActiveCell.AddComment "This is a comment".

The list of methods that an object can perform depends on the object. For example, the Range object supports about 80 different methods. A method is accessed by following the relevant object with a dot and then the name of the method.

*Properties*

An object can have properties. A property is a quality or characteristic of the object, e.g. the length of the dog's tail, the loudness of its bark. If you think of objects as the nouns of VB, then properties are its adjectives and methods are its verbs.

In Excel the properties may themselves be either primitive data types such as numbers, strings or Boolean values, or may themselves be objects of some kind. One of the many properties of the Application object is called Name, and is a read only string containing the name of the application, i.e. "Microsoft Excel". It can be accessed by adding a dot and the Name property after the object:

MsgBox Application.Name

The Application object also has a property called ActiveCell, which represents the currently active cell in the active Excel worksheet. ActiveCell is an instance of the object type called Range, and one of its properties is called Value and represents the value (number, string or formula) held by the cell.

The statement

Application.ActiveCell.Value = "Hello"

will place the string "Hello" in the active cell.

Many properties of the Application object can be used without using the qualifier - Application in this example - and the above statement could simply be written

ActiveCell.Value = "Hello"

Let's consider in detail the statement

Worksheets("sheet1").Range("A1").Value = 3

***Step 1:*** Worksheets("sheet1") evaluates the worksheets method and returns the object that refers to sheet1. So we now have [Worksheet object that refers to sheet1].Range("A1").Value = 3.

***Step 2:*** Evaluates the Range method to return the object that refers to cell A1 of sheet1. So we now have [Range object that refers to cell A1 of sheet1].Value = 3.

***Step 3:*** Evaluates the Value property of the range object at sets it to the number 3.Object variables You have already come across several types of variable in VB, including numerical types (e.g. Integer, Single, Double), the String type, and the Variant type which can hold a value of any type. There is also another type of variable, called an Object variable, which can refer to any of the objects in Visual Basic or in the Excel object model. The syntax for declaring an object variable is much the same as that for any other variable, e.g. Dim rangeA as Range reserves space for a variable that will refer to an object of type Range. One important difference between an object variable and any other type of variable is that an object variable holds only a reference to a specific object, rather than the object itself. (The difference between references and values was discussed in the lecture on modules and procedures, in the context of passing arguments to subroutines and functions.) This distinction may be clearer if we consider a concrete example using primitive and object variables:

```
Dim numA As Integer, numB As Integer
numA = 1
numB = numA
numB = 2
```

```
MsgBox ("A=" & numA & ", B=" & numB)
```

numB is a copy of numA, so setting numB to have the value 2 has no effect on numA, which still has the value 1.

The situation is different for object variables:

```
Dim fontA As Font, fontB As Font
Set fontA = ActiveSheet.Range("A1").Font
fontA.Bold = False
Set fontB = fontA 'Note: fontB and fontA refer to same object
fontB.Bold = True 'so changing object fontB changes object
fontA
```

The object referred to by the variable fontA represents the font used to display the contents of cell A1. fontB is a reference to the same object, not a copy of it, so when the Bold attribute of fontB is changed, fontA is also affected. You may also have noticed the use of the Set keyword in the above examples. This is another difference between object and primitive variables: when assigning an object reference to an object variable, Set must be used, while it is not needed when assigning a value to a primitive variable.

## 5.3 COLLECTIONS

A collection is an object that contains a group of related objects. Each object within the collection is called an element of the collection. Collections are objects so have associated methods and properties.

An example is the Sheets collection, which represents the worksheets in the active workbook. This behaves a bit like an array, in that a specific worksheet in the collection can be referenced using a numeric index:

Sheets(2).Activate

This makes the second worksheet active. Unlike a normal array, the index in a collection object can be a name instead of a number:

Sheets("Chart1").Activate

When you want to work with a single object you usually return one member from the collection. The property or method used to return the object is called an assessor.

There are some useful ways of dealing with collections built into the VB language. For example, to loop over all the members of a particular collection, one can use the For Each syntax:

```
Dim rangeX As Range, cellY As Range
Dim i As Integer
Set rangeX = ActiveSheet.Range("A1:C3")
i = 1
For Each cellY In rangeX.Cells
cellY.Value = i
i = i + 1
```

```
Next
```

The above piece of code uses a loop, in which the object variable cellY refers to each cell member of the collection rangeX.Cells in turn. It assigns the value 1 to A1, 2 to A2, 3 to A3, 4 to B1 etc and to 9 to C3. In this case, we could simply have written rangeX instead of rangeX.Cells, but see what happens if you change it to For Each cellY In rangeX.Rows. Now RangeX represents the same range of cells in each case, but rangeX.Cells and rangeX.Rows represent two different collections of objects. In the first case, the members are individual cells, but in the second case the members are ranges representing rows of cells, with each row being itself a collection with its own members (the individual cells).

### *The With Statement*

The With statement provides a way to carry out several operations on the same object with less typing, and often leads to code that is easier to read and understand. For example, instead of

```
Selection.Font.Name = "Times New Roman"
Selection.Font.FontStyle = "Bold"
Selection.Font.Size = 12
Selection.Font.ColorIndex = 3
```

one can write

```
With Selection.Font
    .Name = "Times New Roman"
    .FontStyle = "Bold"
    .Size = 12
    .ColorIndex = 3
End With
```

### *Using the Macro Recorder to Build an Expression*

The macro recorder is a convenient way to build expressions that return objects as it knows the object model of the application and the methods and properties of the objects. However it can produce very verbose code. Consider the following example to change the size and font in a chart's title:

```
Sub Macro1()
ActivateChart.ChartTitle.Select
With Selection.Font
    .Name = "Times New Roman"
    .FontStyle = "Bold"
    .Size = 24
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = False
```

```
    .ColorIndex = xlAutomatic
    .Background = xlAutomatic
End With
End Sub
```

This code contains many redundant lines. Only the size and fontstyle were changed from the default values. This code after recording should be changed to

```
Sub FormatChartTitle()
With Charts("Chart1").ChartTitle.Font
    .FontStyle = "Bold"
    .Size = 24
End With
End Sub
```

### 5.3.1 Looping on Collections

One can loop over collections using variants of the above example. However the recommended way is to use the For Each ... Next loop. In this structure VB automatically sets an object variable to return, in turn, each object in the collection. The following code loops over all workbooks open in Excel and closes all except the one containing the procedure:

```
Sub CloseWorkbooks()
Dim wb As Workbook
For Each wb In Application.Workbooks
If wb.Name <> ThisWorkbook.Name Then
wb.Close
End If
Next wb
End Sub
```

#### *Range Object and Method*

The range object can represent a single cell, a range of cells, an entire row or column even a 3-D range. The range object is unusual in representing both a single cell and multiple cells.

One of the most common ways to return a range object is to use the range method. The argument to the range method is a string, e.g. A1 or a name "myRange".

Examples

```
Range("B1").Formula = "=10*RAND()"
Range("C1:E3").Value = 6
Range("A1", "E3").ClearContents
Range("myRange").Font.Bold = True
Set newRange = Range("myRange")
```

### *Cells Method*

The cells method is similar to the range method except that it take s numeric arguments instead of string arguments. When used to return a single cell the first argument is the row, the second is the column.

For example, for cell B1, Cells(1,2).Formula = "=10*RAND()"

The cells method is particularly useful if you want to refer to cells in a loop using counters. The *example loops through the cells A1:D10 and replaces any whose value is less than 0.01 by zero.*

```
Sub SetToZero()
For colIndex = 1 To 4
For rowIndex = 1 To 10
If Worksheets("Sheet1").Cells(rowIndex,ColIndex) <_
0.01 Then
Worksheets("Sheet1").Cells(rowIndex, _
colIndex).Value = 0
End If
Next rowIndex
Next colIndex
End Sub
```

### *Combining Range and Cell Method*

You can combine the range and cell methods. Suppose you want to create a range object defined by a top row, a bottom row and left and right columns. The code below returns a range object that refers to cells A1:D10. The cells method defines cells A1 and D10 and the range method then returns the object defined by these cells:

```
Set areaObject = _
Worksheets("Sheet1").Range(Cells(1,1), Cells(10,4))
```

### *The Offset Method*

Sometimes you want to return a range that is a certain number of rows and columns from another cell. The Offset method takes an input range object, with rowoffset and columnoffset arguments to return a range. Suppose the data in a column of cells is either a number or text. The following code writes "Text" or "Number" in the adjacent column.

```
For Each c in Worksheets("sheet1").Range("A1:A10").Cells
If Application.IsText(c.Value) Then
c.Offset(0, 1).Formula = "Text"
ElseIf Application.IsNumber(c.Value) Then
c.Offset(0, 1).Formula = "Number"
End If
Next c
```

### 5.3.2 Manipulating Object Built into Visual Basic

In this we will see that how we Convert an Object to Another Type in Visual Basic. You convert an Object variable to another data type by using a conversion keyword such as CType Function.

Example:

The following example converts an Object variable to an Integer and a String.

```
Public Sub objectConversion(ByVal anObject As Object)
Dim anInteger As Integer
Dim aString As String
anInteger = CType(anObject, Integer)
aString = CType(anObject, String)
End Sub
```

If you know that the contents of an Object variable are of a particular data type, it is better to convert the variable to that data type. If you continue to use the Object variable, you incur either boxing and unboxing (for a value type) or late binding (for a reference type). These operations all take extra execution time and make your performance slower.

Compiling the Code

This example requires:

A reference to the System namespace

## 5.4 CREATING AN OBJECT IN VISUAL BASIC

Object-oriented programming consists of arranging data in modular sets of elements of real world information (called a domain). These data elements are called objects. This data is grouped according to the main real world characteristics of those elements (size, colour, etc...).

The object approach is an idea that has been well proven. Simula was the first programming language to implement the concept of classes in 1967! In 1976, Smalltalk implemented the concepts of encapsulation, aggregation and inheritance (the principle concepts of the object-oriented programming). On the other hand, several object-oriented programming languages have been implemented on a global scale (Eiffel, Objective C, Loops, etc.).

The difficulty with this modular approach is the creation of an abstract representation, in the form of objects, entities that actually exist (dog, car, lightbulb...) or virtually exist (social security, weather, etc.).

An object is characterised by several concepts:

*Attributes:* This is the data that characterises an object. These are variables which store data relating to the state of an object.

*Methods (often called member functions):* An object's methods characterise its behaviour, meaning all actions (called operations) that the object itself is capable of performing. These operations enable the object to respond to external requests (or to act on other objects). Furthermore, operations are closely linked to attributes, as their actions may depend on or even modify attribute values.

***Identity:*** The object has an identity, which distinguishes it from other objects, regardless of its state. This identity is usually created using an identifier which is derived from the type of item (for example: a product may be denoted by a code, a car by a model number, etc.)

## 5.5 BUILDING CLASSES

Classes are a combination of subroutines and types, mean that when you write a class it looks very much like a type but is also contains subroutines and functions.

Each class in VB6 is a distinct file, there can only be one class in a file and the class cannot be spread out in several files; this is a feature of VB6 not a requirement of object oriented programming. A class looks very much like an ordinary module but has the file extension .cls instead of.bas

We can also say that.....

A class is the structure of an object, meaning the definition of all items that an object is made up of. An object is therefore the "result" of a class. In reality, an object is an instance of a class, which is why can use interchange the terms object or instance (or even occurrence).

A class is made up of two parts:

***Attributes*** (often called member data): This is the data that refers to the object's state.

***Methods*** (often referred to as member functions): These are functions that can be applied to objects.

If we have a class called car, the objects Peugeot 406 and Renault 18 will be instances of this class. There may also be other objects Peugeot 406, differentiated by their model number. Furthermore: two instances of a class may have the same attributes but may be considered separate distinct objects. In a real world scenario: two T-shirts may be identical, but they are however distinct from each other. However, if we mix them together it is impossible to distinguish one from the other.

In Visual Basic we can declare two classes dog and cat like this:

```
Class cDog
Option Explicit
    Public Sub Sound()
      Debug.Print "Woof"
End Sub
Class cCat
Option Explicit
    Public Sub Sound()
       Debug.Print "Meow"
End Sub
module main
Public Sub main()
    Dim oDog as cDog
    Dim oCat as cCat
    Set oDog = New cDog
```

```
    Set oCat = New cCat
    oDog.Sound
    oCat.Sound
End Sub
```

When you run this program it will print:

Woof

Meow

**Check Your Progress**

Fill in the blanks:

1. Every object used in Visual Basic has its own ..... to distinguish it from other objects.
2. The Object Browser allows us to browse through the various ....., events and methods.
3. A collection is an object that contains a group of related .....
4. Classes are a combination of ..... and types, mean that when you write a class it looks very much like a type but is also contains subroutines and functions.

## 5.6 LET US SUM UP

Generally, objects in any language supporting OOPs are the variables belonging to a particular user defined data type. These user defined data types are known as Classes. In Visual Basic, the controls on the toolbox represent individual classes and, are called as objects, when placed on a form. The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2. Each object within the collection is called an element of the collection. Collections are objects so have associated methods and properties. Object-oriented programming consists of arranging data in modular sets of elements of real world information (called a domain). These data elements are called objects. This data is grouped according to the main real world characteristics of those elements (size, colour, etc...). Each class in VB6 is a distinct file, there can only be one class in a file and the class cannot be spread out in several files; this is a feature of VB6 not a requirement of object oriented programming.

## 5.7 KEYWORDS

*Event:* An action initiated by the user, the operating system, or the program itself. Examples of events are a keystroke, a mouse click, the expiration of a specified amount of time, or the receipt of data from a port.

*Methods:* Predefined actions that can be performed by an object. For example, a form has a Hide method that makes it invisible to the user.

***Procedures:*** Segments of code that you write to accomplish a task. Procedures are often written to respond to a specific event. Types of procedures include Sub procedures, which consist of a sequence of statements; and Functions, which return a value.

***Properties:*** The characteristics of an object, such as its size, position, color, or text font. Properties determine the appearance and sometimes the behavior of an object. Properties are also used to provide data to an object and to retrieve information from the object.

## 5.8 QUESTIONS FOR DISCUSSION

1. What is object brower? Explain the general object variables.
2. Discuss the manipulation of objects in visual basics.
3. Describe collections. Explain looping on collections.
4. What is the processor to create an object in visual basics?
5. Explain the way of building the classes.

**Check Your Progress: Model Answers**

1. class name
2. properties
3. Objects
4. subroutines

## 5.9 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# LESSON

# 6

# FILES

CONTENTS



## 6.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of files
- Discuss how to identify the sequential files
- Describe the significance of random access files
- Identify and explain the binary files
- Discuss the various sharing files

## 6.1 INTRODUCTION

In many applications, it is helpful to have the capability to read and write information to a disk file. This information could be some computed data or perhaps information loaded into a Visual Basic object.

Visual Basic supports two primary file formats: sequential and random access.

## 6.2 SEQUENTIAL FILE

A sequential file is a line-by-line list of data. You can view a sequential file with any text editor. When using sequential files, you must know the order in which information was written to the file to allow proper reading of the file.

Sequential files can handle both text data and variable values. Sequential access is best when dealing with files that have lines with mixed information of different lengths.

*Sequential File Output (Variables)*

We first look at writing values of variables to sequential files. The first step is to Open a file to write information to. The syntax for opening a sequential file for output is:

Open SeqFileName For Output As #N

Where SeqFileName is the name of the file to open and N is an integer file number. The filename must be a complete path to the file.

When done writing to the file, Close it using:

Close N

Once a file is closed, it is saved on the disk under the path and filename used to open the file.

Information is written to a sequential file one line at a time. Each line of output requires a separate basic statement.

There are two ways to write variables to a sequential file. The first uses the Write statement:

Write #N, [variable list]

Where the variable list has variable names delimited by commas. (If the variable list is omitted, a blank line is printed to the file.) This statement will write one line of information to the file, that line containing the variables specified in the variable List. The variables will be delimited by commas and any string variables will be enclosed in quotes. This is a good format for exporting files to other applications like Excel.

*Example*

```
Dim A As Integer, B As String, C As Single, D As Integer
.
.
Open TestOut For Output As #1
Write #1, A, B, C
Write #1, D
Close 1
```

After this code runs, the file TestOut will have two lines. The first will have the variables A, B, and C, delimited by commas, with B (a string variable) in quotes. The second line will simply have the value of the variable D.

The second way to write variables to a sequential file is with the Print statement:

Print #N, [variable list]

This statement will write one line of information to the file, that line containing the variables specified in the variable list. (If the variable list is omitted, a blank line will be printed.) If the variables in the list are separated with semicolons (;), they are printed with a single space between them in the file. If separated by commas (,), they are spaced in wide columns. Be careful using the Print statement with string variables. The Print statement does not enclose string variables in quotes, hence, when you read such a variable back in, Visual Basic may have trouble knowing where a string ends and begins. It's good practice to 'tack on' quotes to string variables when using Print.

*Example*

```
Dim A As Integer, B As String, C As Single, D As Integer
.
.
Open TestOut For Output As #1
Print #1, A; Chr(34) + B + Chr(34), C
Print #1, D
Close 1
```

After this code runs, the file TestOut will have two lines. The first will have the variables A, B, and C, delimited by spaces. B will be enclosed by quotes [Chr(34)]. The second line will simply have the value of the variable D.

*Quick Example: Writing Variables to Sequential Files*

1. Start a new project.
2. Attach the following code to the Form_Load procedure. This code simply writes a few variables to sequential files.

```
Private Sub Form_Load()
Dim A As Integer, B As String, C As Single, D As Integer
A = 5
B = "Visual Basic"
C = 2.15
D = -20
Open "Test1.Txt" For Output As #1
Open "Test2.Txt" For Output As #2
Write #1, A, B, C
Write #1, D
Print #2, A, B, C
Print #2, D
```

```
Close 1
Close 2
End Sub
```

3. Run the program. Use a text editor (try the Windows 95 Notepad) to examine the contents of the two files, Test1.Txt and Test2.Txt. They are probably in the Visual Basic main directory. Note the difference in the two files, especially how the variables are delimited and the fact that the string variable is not enclosed in quotes in Test2.Txt. Save the application, if you want to.

### *Sequential File Input (Variables)*

To read variables from a sequential file, we essentially reverse the write procedure. First, open the file using:

Open SeqFileName For Input As #N

where N is an integer file number and SeqFileName is a complete file path. The file is closed using:

Close N

The Input statement is used to read in variables from a sequential file. The format is:

Input #N, [variable list]

The variable names in the list are separated by commas. If no variables are listed, the current line in the file N is skipped.

Note variables must be read in exactly the same manner as they were written. So, using our previous example with the variables A, B, C, and D, the appropriate statements are:

Input #1, A, B, C

Input #1, D

These two lines read the variables A, B, and C from the first line in the file and D from the second line. It doesn't matter whether the data was originally written to the file using Write or Print (i.e. commas are ignored).

### *Quick Example: Reading Variables from Sequential Files*

1. Start a new project or simply modify the previous quick example.
2. Attach the following code to the Form_Load procedure. This code reads in files created in the last quick example.

```
Private Sub Form_Load()
Dim A As Integer, B As String, C As Single, D As Integer
Open "Test1.Txt" For Input As #1
Input #1, A, B, C
Debug.Print "A="; A
Debug.Print "B="; B
Debug.Print "C="; C
Input #1, D
```

```
Debug.Print "D="; D
Close 1
End Sub
```

*Note* the Debug.Print statements and how you can add some identifiers (in quotes) for printed information.

3. Run the program. Look in the debug window and note the variable values. Save the application, if you want to.
4. Rerun the program using Test2.Txt as in the input file. What differences do you see? Do you see the problem with using Print and string variables? Because of this problem, I almost always use Write (instead of Print) for saving variable information to files. Edit the Test2.Txt file (in Notepad), putting quotes around the words Visual Basic. Rerun the program using this file as input - it should work fine now.

### *Writing and Reading Text using Sequential Files*

In many applications, we would like to be able to save text information and retrieve it for later reference. This information could be a text file created by an application or the contents of a Visual Basic text box.

### *Writing Text Files*

To write a sequential text file, we follow the simple procedure: open the file, write the file, close the file. If the file is a line-by-line text file, each line of the file is written to disk using a single Print statement:

Print #N, Line

where Line is the current line (a text string). This statement should be in a loop that encompasses all lines of the file. You must know the number of lines in your file, beforehand.

If we want to write the contents of the Text property of a text box named txtExample to a file, we use:

Print #N, txtExample.Text

### *Example*

We have a text box named txtExample. We want to save the contents of the Text property of that box in a file named MyText.ned on the c: drive in the \MyFiles directory. The code to do this is:

```
Open "c:\MyFiles\MyText.ned" For Output As #1
Print #1, txtExample.Text
Close 1
```

The text is now saved in the file for later retrieval.

### *Reading Text Files*

To read the contents of a previously-saved text file, we follow similar steps to the writing process: open the file, read the file, close the file. If the file is a text file, we read each individual line with the Line Input command:

Line Input #1, Line

This line is usually placed in a Do/Loop structure that is repeated untill all lines of the file are read in. The EOF() function can be used to detect an end-of-file condition, if you don't know, a prioiri, how many lines are in the file.

To place the contents of a file opened with number N into the Text property of a text box named txtExample we use the Input function:

txtExample.Text = Input(LOF(N), N)

This Input function has two arguments: LOF(N), the length of the file opened as N and N, the file number.

*Example*

We have a file named MyText.ned stored on the c: drive in the \MyFiles directory. We want to read that text file into the text property of a text box named txtExample. The code to do this is:

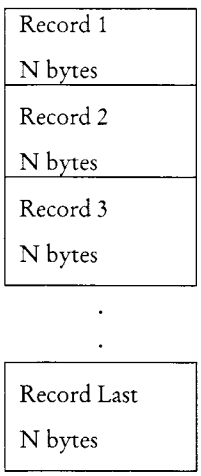
```
Open "c:\MyFiles\MyText.ned" For Input As #1
txtExample.Text = Input(LOF(1), 1)
Close 1
```

The text in the file will now be displayed in the text box.

## 6.3 RANDOM ACCESS FILES

Note that to access a particular data item in a sequential file, you need to read in all items in the file prior to the item of interest. This works acceptably well for small data files of unstructured data, but for large, structured files, this process is time-consuming and wasteful. Sometimes, we need to access data in non-sequential ways. Files which allow non-sequential access are random access files.

To allow non-sequential access to information, a random access file has a very definite structure. A random access file is made up of a number of records, each record having the same length (measured in bytes). Hence, by knowing the length of each record, we can easily determine (or the computer can) where each record begins. The first record in a random access file is Record 1, not 0 as used in Visual Basic arrays. Each record is usually a set of variables, of different types, describing some item. The structure of a random access file is:



A good analogy to illustrate the differences between sequential files and random access files are cassette music tapes and compact discs. To hear a song on a tape (a sequential device), you must go past all songs prior to your selection. To hear a song on a CD (a random access device), you simply go directly to the desired selection. One difference here though is we require all of our random access records to be the same length - not a good choice on CD's!

To write and read random access files, we must know the record length in bytes. Some variable types and their length in bytes are:

Type Length (Bytes)

Integer 2

Long 4

Single 4

Double 8

String 1 byte per character

So, for every variable that is in a file's record, we need to add up the individual variable lengths to obtain the total record length. To ease this task, we introduce the idea of user-defined variables.

*User-Defined Variables*

Data used with random access files is most often stored in user-defined variables. These data types group variables of different types into one assembly with a single, user-defined type associated with the group. Such types significantly simplify the use of random access files.

The Visual Basic keyword Type signals the beginning of a user-defined type declaration and the words End Type signal the end. An example best illustrates establishing a user-defined variable. Say we want to use a variable that describes people by their name, their city, their height, and their weight. We would define a variable of Type Person as follows:

```
Type Person
Name As String
City As String
Height As Integer
Weight As Integer
End Type
```

These variable declarations go in the same code areas as normal variable declarations, depending on desired scope. At this point, we have not reserved any storage for the data. We have simply described to Visual Basic the layout of the data.

To create variables with this newly defined type, we employ the usual Dim statement. For our Person example, we would use:

Dim Lou As Person

Dim John As Person

Dim Mary As Person

And now, we have three variables, each containing all the components of the variable type Person. To refer to a single component within a user-defined type, we use the dot-notation:

VarName.Component

As an example, to obtain Lou's Age, we use:

```
Dim AgeValue as Integer
.
.
AgeValue = Lou.Age
```

Note the similarity to dot-notation we've been using to set properties of various Visual Basic tools.

Writing and Reading Random Access Files

We look at writing and reading random access files using a user-defined variable. For other variable types, refer to Visual Basic on-line help. To open a random access file named RanFileName, use:

Open RanFileName For Random As #N Len = RecordLength

where N is an available file number and RecordLength is the length of each record. Note you don't have to specify an input or output mode. With random access files, as long as they're open, you can write or read to them.

To close a random access file, use:

Close N

As mentioned previously, the record length is the sum of the lengths of all variables that make up a record. A problem arises with String type variables. You don't know their lengths ahead of time. To solve this problem, Visual Basic lets you declare fixed lengths for strings. This allows you to determine record length. If we have a string variable named StrExample we want to limit to 14 characters, we use the declaration:

Dim StrExample As String * 14

Recall each character in a string uses 1 byte, so the length of such a variable is 14 bytes.

Recall our example user-defined variable type, Person. Let's revisit it, now with restricted string lengths:

```
Type Person
Name As String * 40
City As String * 35
Height As Integer
Weight As Integer
End Type
```

The record length for this variable type is 79 bytes (40 + 35 +2 + 2). To open a file named PersonData as File #1, with such records, we would use the statement:

Open PersonData For Random As #1 Len = 79

The Get and Put statements are used to read from and write to random access files, respectively. These statements read or write one record at a time. The syntax for these statements is simple:

Get #N, [RecordNumber], variable

Put #N, [RecordNumber], variable

The Get statement reads from the file and stores data in the variable, whereas the Put statement writes the contents of the specified variable to the file. In each case, you can optionally specify the record number. If you do not specify a record number, the next sequential position is used.

The variable argument in the Get and Put statements is usually a single user defined variable. Once read in, you obtain the component parts of this variable using dot-notation. Prior to writing a user-defined variable to a random access file, you 'load' the component parts using the same dot-notation.

There's a lot more to using random access files; we've only looked at the basics. Refer to your Visual Basic documentation and on-line help for further information. In particular, you need to do a little cute programming when deleting records from a random access file or when 'resorting' records.

### 6.3.1 Using the Open and Save Common Dialog Boxes

Note to both write and read sequential and random access files, we need a file name for the Open statement. To ensure accuracy and completeness, it is suggested that common dialog boxes be used to get this file name information from the user. I'll provide you with a couple of code segments that do just that. Both segments assume you have a common dialog box on your form named cdlFiles, with the CancelError property set equal to True. With this property True, an error is generated by Visual Basic when the user presses the Cancel button in the dialog box. By trapping this error, it allows an elegant exit from the dialog box when canceling the operation is desired.

The code segment to obtain a file name (MyFileName with default extension Ext) for opening a file to read is:

```
Dim MyFileName As String, Ext As String.
cdlFiles.Filter = "Files (*." + Ext + ")|*." + Ext
cdlFiles.DefaultExt = Ext
cdlFiles.DialogTitle = "Open File"
cdlFiles.Flags = cdlOFNFileMustExist + cdlOFNPathMustExist
On Error GoTo No_Open
cdlFiles.ShowOpen
MyFileName = cdlFiles.filename
.
.
Exit Sub
No_Open:
Resume ExitLIne
ExitLine:
Exit Sub
End Sub
```

A few words on what's going on here. First, some properties are set such that only files with Ext (a three letter string variable) extensions are displayed (Filter property), the default extension is Ext (DefaultExt property), the title bar is set (DialogTitle property), and some Flags are set to insure the file and path exist. Error trapping is enabled to trap the Cancel button. Finally, the common dialog box is displayed and the filename property returns with the desired name. That name is put in the string variable MyFileName. What you do after obtaining the file name depends on what type of file you are dealing with. For sequential files, you would open the file, read in the information, and close the file. For random access files, we just open the file here. Reading and writing to/from the file would be handled elsewhere in your coding.

The code segment to retrieve a file name (MyFileName) for writing a file is:

```
Dim MyFileName As String, Ext As String.
cdlFiles.Filter = "Files (*." + Ext + ")|*." + Ext
cdlFiles.DefaultExt = Ext
cdlFiles.DialogTitle = "Save File"
cdlFiles.Flags = cdlOFNOverwritePrompt + cdlOFNPathMustExist
On Error GoTo No_Save
cdlFiles.ShowSave
MyFileName = cdlFiles.filename
.
.
Exit Sub
No_Save:
Resume ExitLine
ExitLine:
Exit Sub
End Sub
```

Note this code is essentially the same used for an Open file name. The Flags property differs slightly. The user is prompted if a previously saved file is selected for overwrite. After obtaining a valid file name for a sequential file, we would open the file for output, write the file, and close it. For a random access file, things are trickier. If we want to save the file with the same name we opened it with, we simply close the file. If the name is different, we must open a file (using a different number) with the new name, write the complete random access file, then close it. Like I said, it's trickier.

We use both of these code segments in the final example where we write and read sequential files.

## 6.4 BINARY FILES

The nitty-gritty of writing a Binary file is that you can put the data into the file any way you want. When a binary file is opened the file pointer is positioned at byte 1. In other words it is at the beginning of the file.

You can write as much as you want into the file while it is open. After each write operation is complete, the file pointer is positioned at the byte immediately after the last data that was written.

For example, if you open the file and write 10 bytes of data, the data begins at byte 1 and ends at byte 10. The pointer is now at byte 11. You can change the pointer in code but that's where it will be if you do nothing. The same thing occurs when reading the file. If you close the file with no further writes to it then the subsequent file will be 10 bytes plus the size of a terminator for each piece of data you stored.

One thing that makes Binary access superior to Random access files is that the string fields can be any length at all. Strings are Visual Basic's largest data type not including objects or User Defined Types. They will quickly eat up memory and disk space, so it's nice to be able to store only as many characters as necessary.

For example, a Random access file must be written using fixed-length records as follows:

```
Option Explicit
Private Type tType
ID As Long
 Name As String * 25
 Address As String * 50
 City As String * 25
 State As String * 2
 ZIP As String * 10
End Type
```

Note that each record will require the full storage size defined for the string fields.

For Binary access we don't need to define the length of the string. If the string is 0 bytes, then 0 bytes get stored (plus some overhead to define the field as a string). The overhead also is used in Random access files, so on that level the files are the same.

```
Option Explicit
Private Type tType
 ID As Long
 Name As String
 Address As String
 City As String
 State As String
 ZIP As String
End Type
```

Now we're not only saving space, but a string field can be longer if needed. So we have two advantages because we are no longer limited by the length of strings.

Writing a Binary File in its Simplest Form

```
Sub WriteBinaryFile ()
Dim i As Integer
Dim nFileNum As Integer
```

```
Dim sFilename As String
sFilename = "C:\Temp\Temp.dat"
' Get an available file number from the system
nFileNum = FreeFile
' Open the file in binary mode. Locks are optional
Open sFilename For Binary Lock Read Write As #nFileNum
 ' Put the data in the file
 For i = 0 To 9
  ' No byte position is specified so writing begins at byte 1
  Put #nFileNum, , i
 Next i
Close #nFileNum
End Sub
' Reading the same file
Sub ReadBinaryFile ()
Dim iValue() As Integer
Dim iCount As Integer
Dim nFileNum As Integer
Dim sFilename As String
Dim x As Integer
sFilename = "C:\Temp\Temp.dat"
' Get an available file number from the system
nFileNum = FreeFile
' Open the file in binary mode. Locks are optional
Open sFilename For Binary Lock Read Write As #nFileNum
 ' Get the data from the file
 Do Until EOF(nFileNum)
  ' Make room in the array for the next value
  Redim Preserve iValue(iCount)
  ' No byte position is specified so reading starts at byte 1
  Get #nFileNum, , iValue(iCount)
  ' Increment counter
  iCount = iCount + 1
 Loop
Close #nFileNum
End Sub
```

You'll notice that when writing the file, we knew how much data was being written, so we could do a For Next loop. However, when reading the file, we don't know so we have to continually

redimension the array. There are a couple ways around this. The first way is to use the FileLen divided by the length of the data. However, that only works if all the data is the same size and type. That happens to be the case here, so we could have done that.

If we want our file to really be flexible then we probably won't be storing data that is always the same size and type. What if we want to store some byte data, a few records, an array of doubles as well as copyright information? We need a way to define what's in the file and the best place to put that information is in the file itself.

For VB DB, I created a header that contains descriptive information. If the file only contained one type of record and nothing else, then there is no need to save the number of records. We would just loop through the file until there was no more data. However, if there is more than one type of data then we need to know when to stop reading a data type and start reading another. This is much easier to accomplish than you might imagine.

The first record stored in the file is the header. Simply create fields in the header to save information about what's in the file.

For example if you wanted to store three different kinds of UDT's, then you would have three **RecordCount** fields. Each field would save the number of records in each array of UDT's.

When the file is read these numbers are extracted from the header and you've got all the information you need to read the rest of the file.

For VB DB, there are three pieces of information stored in the file. We already talked about the header being the first data written. Next are the field definitions.

Field definitions specify what data type each field is as well as other information that should be obvious by looking at the type. The last things stored in the file are the actual records.

## 6.5 SHARING FILES

FileShare Options

| Mode | Description |
|---|---|
| None | The file cannot be opened by any other program until it is closed by the current program |
| Read | Other programs may simultaneously open and read from the file, but not write to it. |
| ReadWrite | Other programs may simultaneously open and read and write from/to the file. |
| Write | Other programs may simultaneously open and write to the file, but not read from it. |

With the above options in mind, the following code excerpt opens 'C:\Temp\text.txt' in*FileMode.OpenOrCreate* with *FileAccess.ReadWrite* permission and no file sharing, and then closes it:

```
Dim textFileStream As New IO.FileStream("C:\Temp\test.txt", IO.FileMode.OpenOrCreate,
IO.FileAccess.ReadWrite, IO.FileShare.None)
textFileStream.Close()
```

Note that the above code example assumes that the 'C:\Temp' directory already exists. If it does not, the code will fail.

**Check Your Progress**

Fill in the blanks:

1. A sequential file is a ..... list of data
2. To open a sequential text file we use .....

## 6.6 LET US SUM UP

In the File System we learn about the concept of file, type of file, and how we deal with tne files. A sequential file is a line-by-line list of data. You can view a sequential file with any text editor. When using sequential files, you must know the order in which information was written to the file to allow proper reading of the file. Note that to access a particular data item in a sequential file, you need to read in all items in the file prior to the item of interest. This works acceptably well for small data files of unstructured data, but for large, structured files, this process is time-consuming and wasteful. Binary file is that you can put the data into the file any way you want. When a binary file is opened the file pointer is positioned at byte 1. In other words it is at the beginning of the file.

## 6.7 KEYWORDS

***Sequential File:*** A sequential file is a line-by-line list of data. You can view a sequential file with any text editor.

***Random Access File:*** Files which allow non-sequential access are random access files.

***Binary File:*** Binary file is that you can put the data into the file any way you want.

## 6.8 QUESTIONS FOR DISCUSSION

1. How sequential files are used to store data in visual basics.
2. What is the difference between sequential file and random access files?
3. Write the code for sharing the files.
4. Discuss the binary files.

**Check Your Progress: Model Answers**

1. line-by-line
2. Print #N, Line

## 6.9 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# UNIT IV

# LESSON

# 7

# COMMUNICATING WITH OTHER WINDOWS APPLICATIONS

CONTENTS



## 7.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of clipboard
- Discuss how to identify the activity window application
- Describe the significance of dynamic data exchange and OLE 2

## 7.1 INTRODUCTION

This lesson covers an introduction to clip board, activity windows application, DDE (Dynamic Data Exchange) & OLE 2.

Application Programming Interface (API) is the interface between an application and the Windows environment. Windows environment has its own unique API. Its programs process user input via messages from the operating system.

## 7.2 CLIP BOARD

The Clipboard object in Visual Basic 6.0 is replaced by a similar Clipboard object in Visual Basic 2005. In Visual Basic 6.0, the Clipboard object is used to store and retrieve text, images, and data to and from

the system Clipboard. Visual Basic 2005 has a Clipboard object in the My Computer namespace with some new methods and some methods that are slightly different.

*Clipboard Data Formats*

In Visual Basic 6.0, constants are provided to define the type of data being passed to or retrieved from the Clipboard. Visual Basic 2005 uses a DataFormats object to define the data type; several new data formats are supported. A list of Visual Basic 6.0 Clipboard format constants and their Visual Basic 2005 equivalents is provided later in this topic.

*Code Changes for the Clipboard Object*

The following example demonstrates storing and retrieving text using the **Clipboard.**

```
' Visual Basic 6.0
Clipboard.Clear
Clipboard.SetText "Hello", vbCFText
If Clipboard.GetFormat(vbCFText) Then
Text1.Text = Clipboard.GetText(vbCFText)
End If
```

The evolution of COM-Component Object Model is closely linked to the need for developing data sharing capabilities across different applications. Before studying COM, let us see and analyze how COM came into existence:

- *ClipBoard:* The earliest form of sharing data across applications. However, in the early days, there were restrictions on the type of data that could be put on the clipboard: lack of graphics support was one such problem.
- *Dynamic Data Exchange (DDE):* The next attempt at sharing data. However, the programming world did not take it too seriously. VB 6.0 provides support for DDE even today, but that is only for maintaining backward-compatibility, i.e., supporting the older applications of VB, which might be using DDE.
- *VBX:* VBX, or Visual Basic Extensions, was introduced with VB 2.0. It was a specification for other developers, in order to develop custom controls in VB. It was 16-bit, though it offered support for both 16-bit and 32-bit programming.
- *Object Linking and Embedding (OLE):* OLE was based on DDE. When we add a picture to an MS-Word document, we are using OLE. When we add a chart from Excel into MS-Word, we are using OLE.
- *OLE2:* This technology was based on COM. It was more successful than OLE in making the programmers interested in its capabilities. However, OLE2 was too bulky. For this reason, it could not become too successful on the Internet, where lightweight applications and technologies are the preferred choices.
- *Active-X:* In order to remove the bulkiness of OLE2, the number of mandatory interfaces to be supported was minimized. This way, Active-X was born. However, Active-X suffers from a serious drawback- it opens the computer to attacks. Active-X components, in order to perform, need to have full access to the system registry. For this reason, Active-X becomes a big security

risk. Consequently, Active-X was not a big success, unlike Java Applets. Java applets offered the advantage of unmatched security, which made them very popular. To fight back, Microsoft went back to the drawing board - rather, marketing board - and came out with COM, which was just a new name for Active-X. Thus, despite all the claims about the freshness of COM, it is basically Active-X at the core. It is really surprising that COM became hugely successful, even though the same technology by the earlier name of Active-X was not successful.

## 7.3 ACTIVITY WINDOWS APPLICATION

- Forms should contain UI-related only; Any logic should live in modules or (even better) class modules
- Class modules make it easier to encapsulate data and code, and reduce the consequences of code changes on the rest of a project
- Use local variables as much as possible, with form- or module-limited variables coming as second best. Avoid project-wide variables as much as possible
- Option Base 1 to reduce the risk of by-one errors in using arrays
- Option Explicit to make sure all the variables are declared before being used
- Indent your code: Since the VB IDE does not support automatic indenting, you can use add-ons such as VB Assistant (a bit buggy, though). Another enhancement to the IDE is MZ-Tools
- Use the Hungarian notation instead of suffixes to make it easier to tell the data type of a variable, eg. sCommand instead of command$
- Use tools like PEBundle to combine the EXE and its DLLs
- Use UPX to compress DLLs and EXEs
- Reduce dependencies as much as possible, eg. hit the Win32 API directly instead of relying on ActiveX controls that perform the same task, use DLL-based embedded SQL engines like SQLite to avoid installing MDAC and ODBC drivers, etc.
- For number-crunching parts, use PowerBasic for Windows (formerly known as PB/DLL) to compile this code into a fast DLL
- Use an installer program such as Inno Setup
- Consider using pointers to increase performance

### 7.3.1 Introduction to API

API stands for Application Programming Interface. The Windows API is a set of several hundred functions and subroutines that are located in a set of files called Dynamic Link Libraries (DLLs). Windows API functions can be simply used just like any other function by declaring the function to be callable from your program. More than 1000 API calls are available and can be classified in four areas:

- Application Manipulation
- Graphics

- System Information
- Registry Interaction

*Dynamic Link Libraries: Windows API Library Files*

The Dynamic Link Library files that make up the Windows API are commonly located in the Windows "System" subdirectory. These files are available on every PC using Windows operating system. The three important Windows DLLs are:

- User32.dll
- GDI32.dll
- Kernel32.dll

The other extension DLLs are also present in the System directory to provide additional functionality besides the required one. Some other useful extension DLLs are:

- Comctl32.dll
- Commdlg.dll
- Dll32.dll
- Version.dll
- Apigid.dll
- Mapi32.dll
- Winmm.dll
- Odbc32.dll
- Netapi32.dll

**User32.DLL:** The User32.DLL library file contains functions that relate to managing the windows environment such as:

- Handling messages between windows
- Managing cursors
- Handling other non-display functions
- Managing menus

**GDI32.DLL:** This file contains functions that help to manage output to different devices, especially the screen. Some important functions are BitBlt&, DeleteObject&, RoundRect&, SelectObject& etc.

**Kernel32.DLL:** The Kernel32.DLL file contains functions that manage the low-level operating system functions. These functions include:

- Memory Management
- Task Management
- Resource Handling
- File and Directory Management
- Module Management

Some useful functions are GetSystemDirectory&, GetTempFileName&, GetModuleFileName& etc.

*Declaring a DLL Function*

A DLL procedure must be declared before it is called. To declare a DLL procedure, use Declare statement to the Declaration section of the code window. If the procedure returns a value use the following syntax:

```
[Public | Private] Declare function Publicname Lib "libname"[Alias "alias"]
[([[ByVal]_
variable [As Type] () [By Val]] variable [As Type]]…])] As Type
```

If the procedure does not return a value use the following syntax:

```
[Public | Private] Declare function Publicname Lib "libname"[Alias "alias"]
[([[ByVal]_
variable [As Type] () [By Val]] variable [As Type]]…])]
```

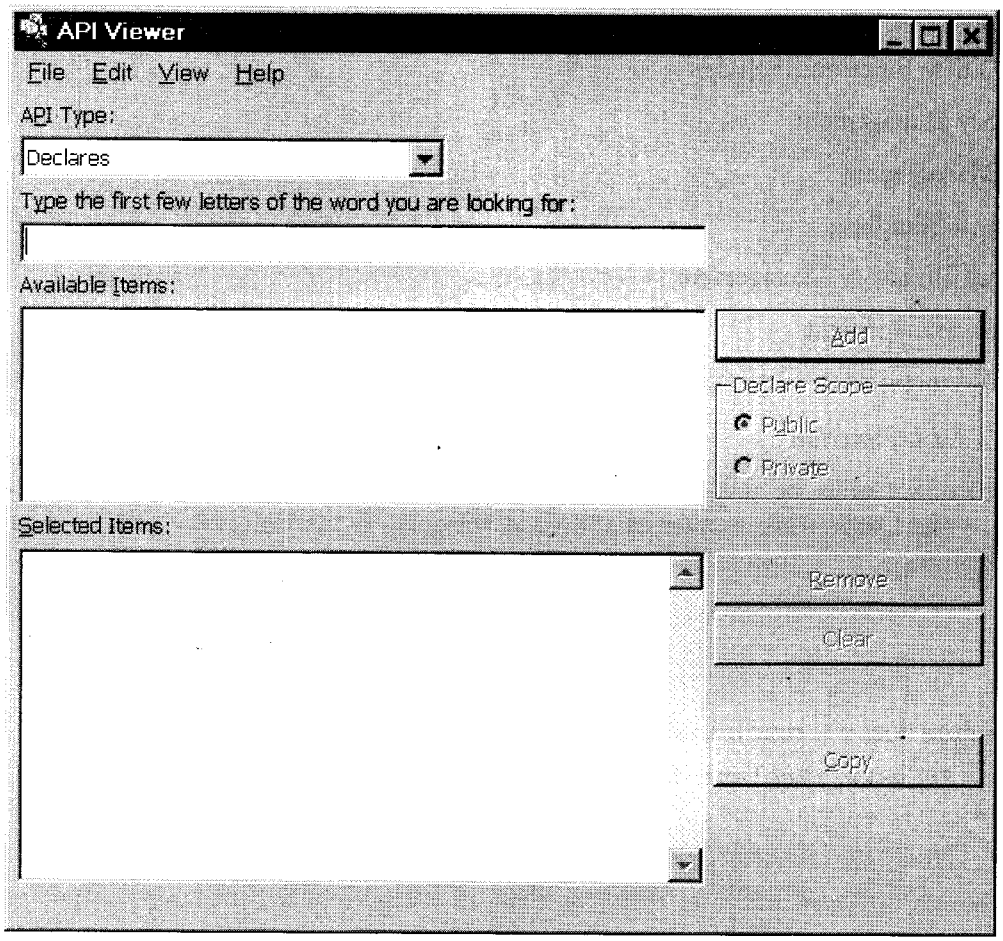
Public, although optional, indicates that you want all modules in the program to have access to this API function. By omitting Public keyword, scope of the functions gets limited to the scope of the declaring section. Alias keyword is to specify an alternative name of the function to be used in the program code. Alias names are used to distinguish API function names from Visual Basic's keywords.

*The API Viewer Application*

The API Viewer Application enables you to browse through the declares, constants and types included in any text file or Microsoft Jet database. After you find the procedure you want, you can copy the code to the file Clipboard and paste it into your Visual Basic application. You can add as many procedures as you want into your application.

*Using the API Viewer Application*

To view an API file from the Add-Ins menu, open the Add-In Manager and Load API Viewer. Click API Viewer from the Add-Ins menu. Open the text or database file you want to view. To load a text file into viewer, Click File\Load Text file and choose the file you want to view. To load a database file, click File\Load Database File.




Select the type of item you want to view from the API Types list. You can have the API Viewer automatically display the last file you viewed in it, when it is opened, by selecting View\Load Last File. To add procedures to your Visual Basic code, click the procedure you want to copy in the Available Items list. Click Add. The item appears in the Selected Items list.

Indicate the scope of the item by clicking Public or Private in the Declare Scope group. To remove an entry from the Selected Items list box, click the item and click Remove. To remove all entries from the Selected Items list box, click Clear. To copy the selected items to the clipboard, click Copy.

All of the items in the Selected Items list will be copied. Open your Visual Basic project and the module in which you want to place the API information. Position the insertion point where you want to paste the declarations, constants, and/or types, and then choose Edit\Paste.

*Using Application Programming Interface*

Visual Basic doubles up its power by extending its access to API. It provides a balance by hiding many of the complexities of windows programming, while still providing access to Windows and its utilities.

It has already been discussed that before using an API function, you need to understand the categories of the functions because each category has a different library. For example, if you need to add basic functions to build and manage the display of your program output and to capture user input, you

must look in User32.dll. Similarly, functions related to graphics can be obtained from GDI32.Dll. Using a DLL procedure in Visual Basic consists of two steps:

- Declaring the API function
- Calling the API function

You have to declare each and every function before using it. The best way to declare a function is to copy it from the API text viewer and then paste it in your module. This prevents invalid declarations of functions.

### *Using The FlashWindow() API Function*

The FlashWindow() in the User32.dll flashes the titlebar of the window whose handle is passed. The FlashWindow() function has the following declaration statement.

```
Declare Function FlashWindow Lib "user32" (ByVal hwnd As_
Long, ByVal Invert As Long) As Long
```

The hwnd argument is the handle of the window and Invert argument specifies whether title bar is to be flashed or not. Non-zero value specifies that the titlebar will not be flashed.

To use this function, declare it in the module in your project and add a Timer control on your form with interval property set to 500. Add a new command button on the form with the caption "Exit". Add the following code in the event handler for Timer event of Timer control.

```
Private Sub Timer1_Timer()
Static Invert
Dim X, hForm1 As Long
X = FlashWindow(hForm1, Invert)
Invert = NotInvert
End Sub
```

Type the following code to exit from the application:

```
Private Sub Command1_Click()
End
End Sub
```

Save the project with API Project name and run it. The titlebar of the window will flash after every ½ second. This function can be used to draw the attention of users while the window is an inactive window.

**Check Your Progress**

Fill in the blanks:

1. The Clipboard object is used to ..... and retrieve text, images, and data
2. The Windows API is a set of several hundred functions and ..... that are located in a set of files called Dynamic Link Libraries (DLLs).

## 7.4 LET US SUM UP

In Visual Basic 6.0, constants are provided to define the type of data being passed to or retrieved from the Clipboard. Visual Basic 2005 uses a DataFormats object to define the data type; several new data formats are supported. A list of Visual Basic 6.0 Clipboard format constants and their Visual Basic 2005 equivalents is provided later in this topic. API stands for Application Programming Interface. The Windows API is a set of several hundred functions and subroutines that are located in a set of files called Dynamic Link Libraries (DLLs). Windows API functions can be simply used just like any other function by declaring the function to be callable from your program.

## 7.5 KEYWORDS

*ClipBoard:* The earliest form of sharing data across applications.

*Dynamic Data Exchange (DDE):* The next attempt at sharing data. However, the programming world did not take it too seriously. VB 6.0 provides support for DDE even today, but that is only for maintaining backward-compatibility, i.e., supporting the older applications of VB, which might be using DDE.

*VBX:* VBX, or Visual Basic Extensions, was introduced with VB 2.0.

*Object Linking and Embedding (OLE):* OLE was based on DDE. When we add a picture to an MS-Word document, we are using OLE. When we add a chart from Excel into MS-Word, we are using OLE.

*OLE2:* This technology was based on COM. It was more successful than OLE in making the programmers interested in its capabilities.

## 7.6 QUESTIONS FOR DISCUSSION

1. For what purpose the clip board is used?
2. Explain the dynamic data exchange.
3. What are the keywords in an API function declaration in Visual Basic?
4. What are Dynamic Link Libraries? How are they different from executable .exe files?

**Check Your Progress: Model Answers**

1. Store
2. subroutines

## 7.7 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# LESSON

# 8

# DATABASE FEATURES

CONTENTS

## 8.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept modern databases
- Discuss how to identify the data manger
- Describe the significance of using the data control
- Identify and explain the programming of data control
- Discuss the changes in data control
- Explain the concept of SQL
- Describe the database objects
- Explain the ADO and OLE.DB

## 8.1 INTRODUCTION

Till this we seen the power of the built-in Visual Basic tools. In this we look at one of the more powerful tools, the Data Control. Using this tool, in conjunction with associated 'data-aware' tools, allows us to access and manage databases. We only introduce the ideas of database access and management - these topics alone could easily take up a ten week course.

A major change in Visual Basic, with the introduction of Version 6.0, is in its database management tools. New tools based on ActiveX Data Object (ADO) technology have been developed. These new tools will eventually replace the older database tools, called DAO (Data Access Object) tools.

## 8.2 MODERN DATABASES

In simplest terms, a database is a collection of information. This collection is stored in well-defined tables, or matrices.

The rows in a database table are used to describe similar items. The rows are referred to as database records. In general, no two rows in a database table will be alike.

The columns in a database table provide characteristics of the records. These characteristics are called database fields. Each field contains one specific piece of information. In defining a database field, you specify the data type, assign a length, and describe other attributes.

***Database Structure and Terminology***

In simplest terms, a database is a collection of information. This collection is stored in well-defined tables, or matrices.

Databases are designed to offer an organized mechanism for storing, managing and retrieving information. They do so through the use of tables. If you're familiar with spreadsheets like Microsoft Excel.

### *Database Tables*

Just like Excel tables, database tables consist of columns and rows. Each column contains a different type of attribute and each row corresponds to a single record.

The rows in a database table are used to describe similar items. The rows are referred to as database records. In general, no two rows in a database table will be alike.

The columns in a database table provide characteristics of the records. These characteristics are called database fields. Each field contains one specific piece of information. In defining a database field, you specify the data type, assign a length, and describe other attributes.

Here is a simple database:

Field Records

| Id No | Name | Date of birth | Height | Weight |
|---|---|---|---|---|
| 1 | Jones | 01/04/58 | 86 | 120 |
| 2 | Rake | 01/02/60 | 60 | 130 |
| 3 | Jam | 02/03/78 | 50 | 145 |
| . | . | . | . | . |
| . | . | . | . | . |

In this database table, each record represents a single individual. The fields (descriptors of the individuals) include an identification number (ID No), Name, Date of Birth, Height, and Weight.

Most databases use indexes to allow faster access to the information in the database. Indexes are sorted lists that point to a particular row in a table. In the example just seen, the ID No field could be used as an index.

A database using a single table is called a flat database. Most databases are made up of many tables. When using multiple tables within a database, these tables must have some common fields to allow cross-referencing of the tables. The referral of one table to another via a common field is called a relation. Such groupings of tables are called relational databases.

In our first example, we will use a sample database that comes with Visual Basic. This database (BIBLIO.MDB) is found in the main Visual Basic directory (tryc:\Program Files\Microsoft Visual Studio\VB98). It is a database of books about computers. Let's look at its relational structure. The BIBLIO.MDB database is made up of four tables:

***Authors Table (6246 Records, 3 Fields)***

| Au_ID | Author | Year Born |
|---|---|---|
| | | |
| | | |

*Publishers Table (727 Records, 10 Fields)*

| Pub_ID | Name | Company | Fax | Comments |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |

*Title Author Table (16056 Records, 2 Fields)*

| ISBN | AU_ID |
|---|---|
| | |
| | |

*Titles Table (8569 Records, 8 Fields)*

| Title | Year Pub | ISBN | PubId | Comments |
|---|---|---|---|---|
| | | | | |
| | | | | |

The Authors table consists of author identification numbers, the author's name, and the year born. The Publishers table has information regarding book publishers. Some of the fields include an identification number, the publisher name, and pertinent phone numbers. The Title Author table correlates a book's ISBN (a universal number assigned to books) with an author's identification number. And, the Titles table has several fields describing each individual book, including title, ISBN, and publisher identification.

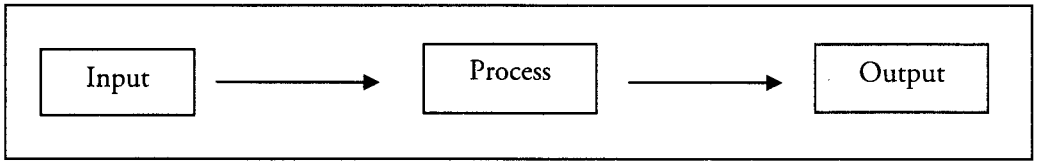
Note each table has two types of information: source data and relational data. Source data is actual information, such as titles and author names. Relational data are references to data in other tables, such as Au_ID and PubID. In the Authors, Publishers and Title Author tables, the first column is used as the table index. In the Titles table, the ISBN value is the index.

Using the relational data in the four tables, we should be able to obtain a complete description of any book title in the database.

We can form alternate tables from a database's inherent tables. Such virtual tables, or logical views, are made using queries of the database. A query is simply a request for information from the database tables. As an example with the BIBLIO.MDB database, using pre-defined query languages, we could 'ask' the database to form a table of all authors and books published after 1992, or provide all author names starting with B. We'll look briefly at queries.

Keeping track of all the information in a database is handled by a Database Management System (DBMS). They are used to create and maintain databases. Examples of commercial DBMS programs are Microsoft Access, Microsoft FoxPro, Borland Paradox, Borland dBase, and Claris FileMaker. We can also use Visual Basic to develop a DBMS. Visual Basic shares the same 'engine' used by Microsoft Access, known as the Jet engine. In this class, we will see how to use Visual Basic to access data, display data, and perform some elementary management operations.

A 'system' is an orderly collection of inter-dependent components and procedures, working collectively for a common goal. Any 'system', whether big or small, involves the following three steps:




As per the working of a general system described in the figure above, the Database System works in the following manner:

1. The input is the data.
2. The data is stored and processed on the user's demand.
3. The output is the information generated from the data.

The database system (dbs) is the sum total of all the components that are involved in the collection, storage and retrieval of data. This way, we can define the dbs as a system that accepts raw data as an input and generates information as the output.

The components of a *dbs* are:

- Data (input, the starting point)
- Hardware (for data storage)
- Software (for processing data)
- Users (for creating/using software for processing data)

The DBS is a computerized record-keeping system.

### *Relational Database*

There are three accepted models for storage and organization of data:

- Hierarchical model
- Network model
- Relational model

A database based on the *relational model* is known as a *relational database*. In a *relational database,* the user perceives (or sees) the data in the form of 'tables'. The word 'table' means a combination of rows and columns, and this is the method in which the user will see the data in this model. Here is a sample representation of a 'table' named 'employee':

Columns ↓ ↓

Rows →

| EmpNo | EmpName | Salary | Dept | Designation |
|---|---|---|---|---|
| 1 | Mannu | 10000 | F | Sr. Mgr |
| 2 | Neenu | 8000 | P | Dy. Mgr |
| 3 | Kuhu | 9000 | P | Mgr |
| 4 | Juju | 8000 | M | Dy. Mgr |
| 5 | Sheenu | 7000 | H | Dy. Mgr |
| 6 | Nimit | 5000 | M | Exec. |

A relational database will include many such tables. Each table will hold data related to a particular object, or 'entity', such as:

- Employee entity (shown on previous page)
- Courses entity
- Students entity
- Faculty entity
- Marks entity, etc.

The columns are the properties of the entity. Through the columns, we can 'describe' an entity. The rows are a combination of columns. Each row represents a unique member of an entity. In a relational database, fields are called columns or attributes. Records are known as rows or tuples. Files are also known as tables or relations or entities.

*Primary Key*

We organize data in a relational database in the form of tables. Over a period of time, we will have a large number of records in our tables. Consider a university with over 1 lakh students. Suppose we want to find details of a particular student, we will have a problem if we specify only the name - after all, there can be many students with the same name and that too in the same class. Thus, keeping track of a particular record/row can be a big problem in a big table.

Suppose we specify our search criteria as the name of a student Saksham Puri, we might end up with 24 students with the same name. Again, we will be stuck. Though we needed just one student's details from over 1 lakh, we will be having 24 possible students to choose from. We are facing a problem of too much information.

To overcome this problem of too much information, we need a method which can help us pin-point a particular row in a table. This mechanism is called as the *primary key*. Thus, a *primary key* is a column (or columns) of a table to uniquely identify a row in it. The *primary key* makes sure that all the rows in a table are distinct in some manner, though the data they have, is closely related.

Now once again consider how records are accessed commonly. We all know that:

- a student's record is only accessed by using 'enrolment number'.
- for a railway reservation, we are always given a 'PNR Number'.

This PNR Number is always used to enquire about our ticket. As we know, our name is not used for checking on any details on our ticket - it is the PNR Number which is all-important. If we have three passengers named Tarush Gupta travelling from Jammu to New Delhi on the same day and train, we can easily identify each passenger uniquely by using the PNR Number. If we use the name to identify each passenger, we will be in big trouble.

A primary key possesses two essential properties:

- It will always be *unique* (distinct).
- It will always be *not null* ('blank' values will not be allowed).

### *Foreign Key*

In a relational database, we deal with multiple tables, not just one. More importantly, these tables will not be totally 'isolated' from one another. They will be connected or *related* to each other in some way or the other - on the basis of some common attribute or *column.* (This is the essence of the word 'relational' in a relational database).

Consider the table of employees 'emp' we had seen some time back in this lesson itself. Now consider another table 'department', with details as follows:

| Dept | Dept_Name | Location | Budget |
|---|---|---|---|
| FI | Finance | New Delhi | 2000 |
| PR | Production | Gurgaon | 5500 |
| MI | MIS | Noida | 3000 |
| HR | HRD | New Delhi | 1500 |

The department table, as we are seeing, contains details of the various departments within the company. The primary key of this table is the column dept. How do we relate tables emp and department? If you guessed the column dept, you are 100% on target! Both the emp and department tables have a common column, dept. This is a common data for them. Hence, this column can become the joining factor for both of these tables.

If we need to see the employees, data alongwith the details of their departments, we can *join* the data of both the tables, based on the common values of dept in both the tables. Wherever the dept in department matches the dept in emp, we can combine the data of both the tables into a single row.

The column dept will also serve another extremely important purpose. The dept is the primary key in department table. In emp table, if we make dept as the *foreign key*, the foreign key will do this for us:

If a user enters a 'dept' code in 'emp' that does not exist in 'department', the DBMS gives an error. For example, if the user types 'dept' as 'MI' or 'FI' or 'HR' (in 'emp' as given above), it will be accepted by the database. If the value of 'dept' is entered as 'TI' (in 'emp' table), it will be rejected. This is because the same value ('TI') does not exist for 'dept' (in 'department' table).

Thus, the *foreign key* of a table is the primary key of some other table. It ensures that column values in the foreign key always match the column values in the corresponding primary key. That is, the *foreign key* compares the values with the values in its 'parent table'. For this reason, the *foreign key* is said to enforce *referential integrity* (enforcing integrity through cross-referencing). When a data is entered into the foreign key, the foreign key 'references' or cross-checks the related table to see if that value already exists. If a match is found, the data is accepted and the row is added to the table. If the match does not exist, the data is rejected by the database and the row addition to the table is aborted.

The primary key is for uniqueness of a row in a single table. The foreign key is for referential integrity across two tables.

### *Normalization*

Let us assume that we are managing a computer training institute. Also assume that we are using a computer to maintain all our data:

- students,

- courses,
- faculties, etc.

In the earlier years, the *flat-file structure* was very popular. Based on this *flat-file structure*, the structure of our data file might be like this:

| Roll_No | Name | Counsellor | Course1 | Desc1 | Faculty1 | Course2 | Desc2 | Faculty2 |
|---|---|---|---|---|---|---|---|---|
| 1 | Himangi | Arjun | V6 | VB6 + COM | Mamta | O8 | Oracle8 OOP | Rakhi |
| 2 | Tananya | Karan | A3 | ASP 3 | Eva | V6 | VB6 + COM | Mamta |
| 3 | Kangna | Karan | O8 | Oracle8 OOP | Rakhi | D | DHTML VBS | Aarti |
| 4 | Shivam | Shivam | A3 | ASP 3 | Eva | O8 | Oracle8 OOP | Rakhi |
| 5 | Tarush | Tarush | V6 | VB6 + COM | Mamta | O8 | Oracle8 OOP | Rakhi |
| 6 | Tamanna | Arjun | D | DHTML VBS | Aarti | E | ERP SAP | Anshu |

*Need*

At first glance, this file appears to provide all the information – students, faculty, and even courses. All the information is available right in front of our eyes, through a simple surfacescan. But, if we forget the ease of data visibility and begin to analyze the design of the file, it begins to fall apart. Let us see the problems here:

1. ***Redundant Data:*** V6, O8 and A3, and their related data is seen at many places. This is obviously a case of repetition of data, i.e., redundancy. We also see repeating groups: Course1, Desc1, Faculty1, Course2, Desc2, Faculty2, and so on. Repeating groups and redundant data are very difficult to manage, as we will just see.
2. ***Data Modification:*** If the course description for V6 changes from [VB6 + COM] to [VB6 with COM/DCOM], we need to make the changes at all the places where V6 has been entered. In a small file it is easy. But suppose our file was big, what then? Wouldn't it be difficult to change the values in, say, 1,500 records? If we miss even a single record in the change process, we will be having inconsistent data–inconsistent data is a big problem to handle.
3. ***Data Insertion:*** If we want to add a new course, how do we add it? As a new repeating group, of course (Course3, Desc3, Faculty3). But where do we add it? After the 'Faculty2' column? Fine, done. But, what about the student name against which we will be adding the details? As per the file, that student will be shown a student of the new course, even if that is not the reality. This will be very unfair for that student; isn't it?
4. ***Data Deletion:*** In the file we see, Tamanna is the only student studying course E (ERP). Suppose she leaves the course and cancels her registration? Well, we will do it – with a pinch of salt. After this record deletion, we cannot say whether our institute teaches ERP or not. This is because the only reference to ERP has been deleted along with the student's name– Tamanna, in this case.

The reason behind these problems is that the file designing has been very bad. *Normalization* is the process of taking a *flat file* and 'breaking it down' into small *tables*, so that the day-to-day maintenance

- insertion, deletion and updation - of the data becomes easy. However, while breaking the file, we should ensure that we don't lose any information contained in it. In other words, whenever needed, we should be able to re-construct the original, *flat view* from our broken-down tables. This is known as *non-loss decomposition* or *lossless decomposition.*

Just think, the end-user is most happy and comfortable seeing the *flat-file* above. However, as database experts, we have just seen that we are not happy with this file design. Thus, there is a peculiar situation here:

- We want to break down a single file into smaller, easily-managed tables : a fragmented view, so to say.
- The end-user wants to see all the data on a single screen : a unified view, opposed to a fragmented view.

This is a real conflict of interests. The end-user wants simplicity in viewing. The designer wants a simplicity in storage. As we have just seen, these are two entirely opposite requirements. Focusing on ease of display affects the design process whereas focusing on proper designing affects the ease of display. *Normalization* offers the best solution to these conflicting needs:

- The data can be stored internally in different, broken-down tables - easy for the database designer.
- When required, the data can be joined and presented in a unified manner - easy for the end-user.

Thus, *normalization* solves the problems of both the designer and the end-user. It is like a bridge between the two conflicting requirements of the end-user and the database designer.

## 8.3 DATABASE FUNDAMENTALS

One of the most important aspects of working in VB is programming for databases. When we say the world is witnessing an information technology revolution, it is important to understand that it is data and databases that are driving it. Database is one of the most important resources of any good company.

One of the reasons behind the huge popularity of VB today is Database Programming. Data is present on a host of environments today:

- On the desktop,
- On a database server in a client-server environment, and
- On the Internet.

Whatever be the environment, VB offers the benefit of hiding a lot of the database-level tasks from the programmers. Thus, VB helps in database implementation with minimum effort. It is quite common to see programmers developing a VB-Oracle application with a very elementary knowledge of Oracle.

Before we start programming for database applications, here is an overview of the basic database concepts. Kindly note that this is just an overview, not a complete review.

## 8.4 DATA MANAGER

Most of the time, we do not know the manner in which the data is maintained internally. For the end-user, the data is always presented in a simple, easy to understand format. All the complex details involved in:

- storing data,
- indexing the data and
- retrieving the data

are hidden from us, the end-users. *Database Management System (DBMS)* is the software that hides all of these aspects from us, making data-handling a fairly easy task. Thus, *dbms* deals with the complex low-level handling of the database, while providing us a simple, easy-to-use interface. This way, the DBMS will offer the following features:

- Creation of storage structures (tables, indexes, views, etc).
- Manipulation of data (insert, update, delete).
- Checking data integrity (validation).
- Making data secure from unwanted users.
- Data recovery ('recovering' from a system failure).
- Managing data concurrency (many users accessing same record).

In a nutshell, we can say that the *DBMS* is the software that is responsible for handling all the issues related to accessing the database. Similar to high-level languages that hide complexity of low-level language, *DBMS* hides low-level data storage details.

## 8.5 USING THE DATA CONTROL

With the Data Control, the implementation details of database-related tasks are hidden from us. The Data Control is based on the DAO model. It offers a simple, easy-to-use interface. Using this, we can use DAO without even understanding DAO programming in-depth. We just set a few important properties and link the Data Control to some controls for displaying or accepting data. It is really simple.

On the control, we have four buttons with arrows marked on them. These buttons perform the vital task of navigation through our records:

1. Moving to the first record (1st left button).
2. Moving to the previous record (2nd left button).
3. Moving to the next record (2nd right button).
4. Moving to the last record (1st right button).

We will work on these buttons for most of the time.

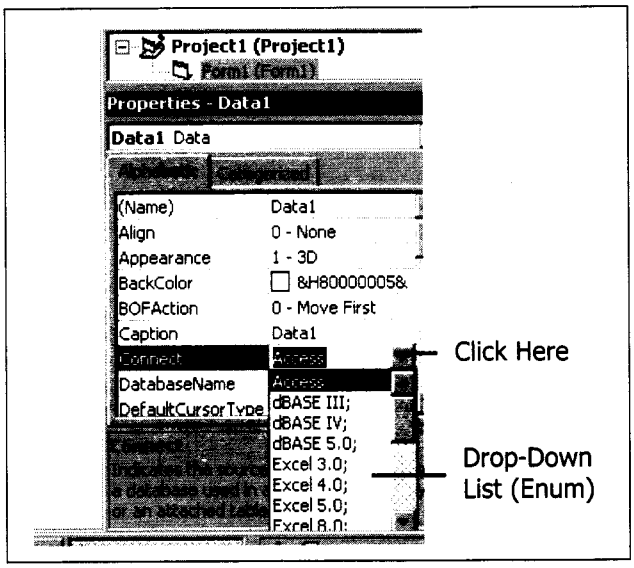
### 8.5.1 Working with the Data Control

We set the following properties for the Data Control:

*Connect: Linking to a Back-end*

This is the first task while working with the Data Control. For specifying the software to use for our application, we use the connect property of the Data Control. This property specifies which software will act as the database (Access, Excel, etc.) To set the connect property, do the following:

1. Select the Data Control and come to the properties window, which now displays the properties for the Data Control.
2. Click on 'connect' property. Observe the drop-down list (enum):



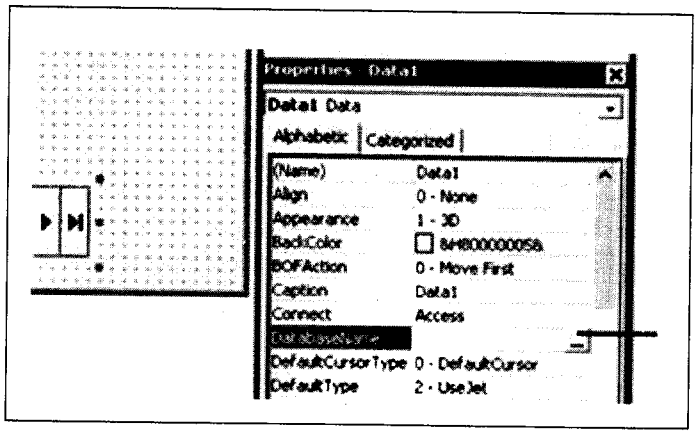

The connect property has now been set. We should be seeing the word 'Access' written next to the property name.

If Access is the back-end, save time by skipping this step, since, Access is the default selection in the list.

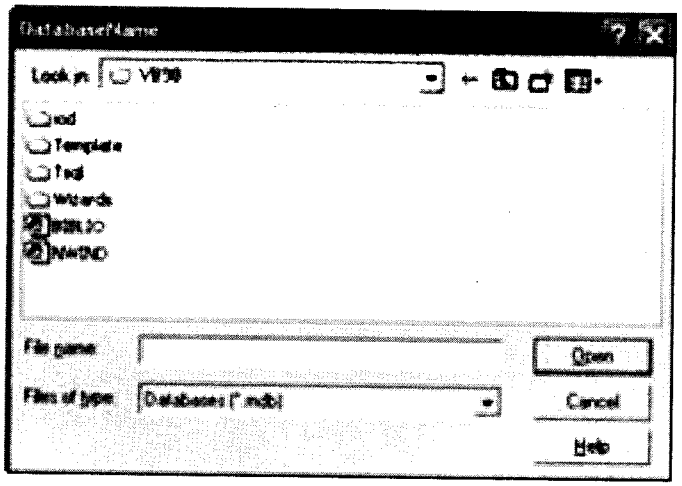
*Database Name: Setting the Database Link*

After connect, we set the DatabaseName property. This property, as is expected, will set the name of the database, i.e., full path. To set the DatabaseName property, we take the following steps:

1. Select the Data Control and click on 'DatabaseName' property in the properties window. An 'ellipsis' appears at the extreme right of the window.

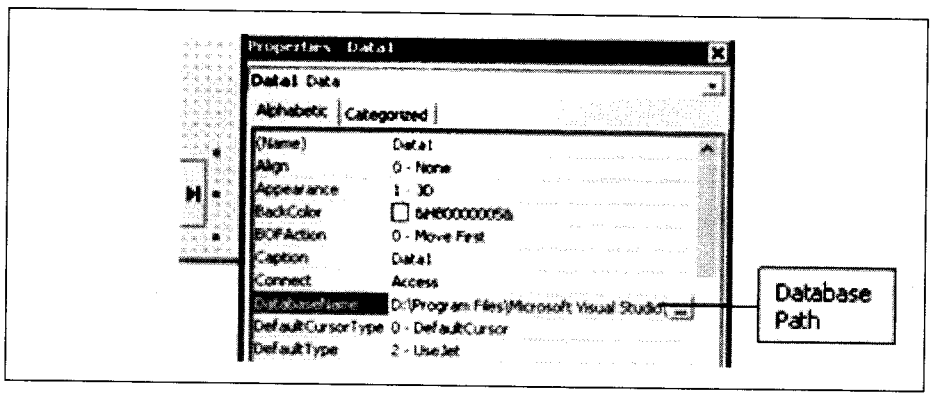



2. Click on the ellipsis, the 'Database Name' dialog box opens up.




3. From this dialog box choose the name of the Access database to work with, and click on 'Open'. The box closes.

   The property DatabaseName has been set. We can see the full path of the database next to the property name.



*Displaying/Entering Data: Binding Controls to Data Control*

The data control is now ready. Now we will link some textboxes to our data control. This way, we can view and type the data through the textboxes. After all, the data control is making the data 'available', but it cannot 'show' the data on its own. To view/type the data, we add as many textboxes and associated labels to the form as is required - 1 textbox per field. Then we set two important properties of the textboxes:

*DataSource:* This property links the textbox to a Data Control. To set this:

1. Select the textbox and click on this property.
2. Click on the drop-down arrow which now appears.
3. From the drop-down list which now appears, choose the appropriate Data Control name.

*DataField:* This property links the textbox to a field (or column) from the table linked to the Data Control (recordsource). To set this:

1. Select the textbox and click on this property.
2. Click on the drop-down arrow which now appears.
3. From the drop-down list which now appears, choose the desired field name.

Once these two properties have been set for all textboxes, we add Labels to the Form. These will act as prompts for data that will be displayed in the textboxes. Add the required number of Labels and give suitable captions.Setting datasource and datafield properties is known as 'binding' or 'data binding'. Our Textboxes are now called 'bound controls' or 'data-bound controls'.

Now run the program. We will see the records from our table being displayed in the textboxes. This 'set of records' which we have retrieved from the database is called as a recordset in DAO.

Thus, in a nutshell:

1. The Data Control makes a recordset available (through the databasename and the recordsource property).
2. This recordset object consists of a collection of fields that can be shown in the respective textboxes.

## 8.6 PROGRAMMING WITH DATA CONTROL

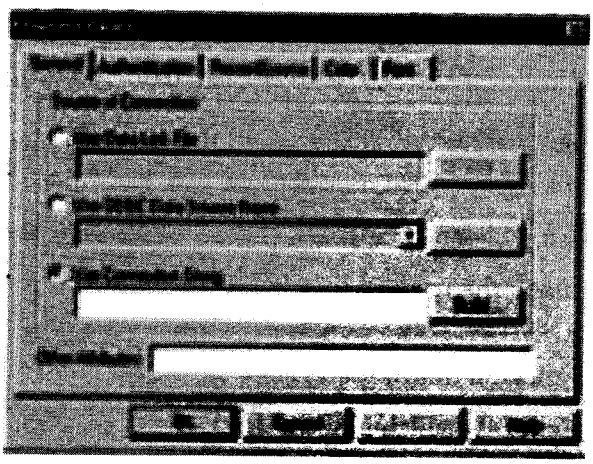
### 8.6.1 ADO Data Control

*Calling on ADO Data Control*

The ADO Data Control (Adodc) is not available in the toolbox by default. To get it into the toolbox, we follow the by-now familiar route of clicking on 'Project' 'Components'. From the dialog box, we select 'Microsoft ADO Data Control 6.0 (OLE DB)'. Though the Adodc looks like the DAO-based Data Control, the internal working is quite different.

### *Setting up Basic Properties*

1. Right-click the Adodc on the form. A pop-up menu appears.
2. Select 'Properties' from the pop-up menu. The property pages for the Adodc open up, as shown below:



On the property pages, two tabs are most important:

- General - helps set the connection object
- RecordSource - helps set command object, to create the recordset

### *Specifying the Connection*

The General Tab helps configure the connection string for the connection object. It offers the following options for setting the connection string:

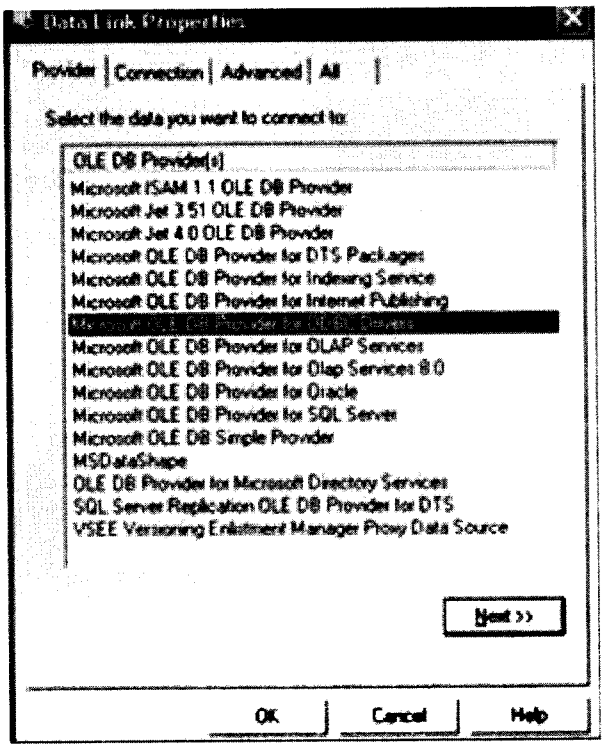
- Select 'Use ODBC Data Source Name' to work through a DSN.
- Select 'Use Connection String' to either use an existing DSN via OLE DB or make a new OLE DB Provider connection.

If we select the 'Use ODBC Data Source' option, we have two choices:

- Select an existing DSN from the drop-down list, or
- Create a new DSN by clicking on 'New'.

The process of creating a DSN has already been covered in the previous chapter's section on ODBC 'Creating a DSN'.

If we select the 'Use Connection String' option and click on 'Build', we will be presented with a screen such as this:




Aha! This is something we have already seen before under 'Setting the DED: (through OLE DB)' section of the DED. For going further from here, refer to the 'DED configuration through OLE DB' section. Since DED and Adodc are both based on ADO technology, the configuration is not much different.

*Specifying the Command*

Once Connection has been set, the Command object needs to be set. The command object will give us the recordset at runtime. For this, we click on the 'RecordSource' tab, where we get the following four options. Their significance is as on next page:

(a) *Command Type:* Mention the type of Command we will use:

| Vbconstant | Meaning |
|---|---|
| Adcmdunknown | Command type is unknown |
| Adcmdtext | Command is an SQL query |
| Adcmdstoredprocedure | Command is a stored procedure |
| Adcmdtable | Command is an SQL table name |

(b) *Command Name:* A drop-down list of table or procedure names from the Connection, which helps create the Recordset.

(c) *SQL Text:* Here we write the SQL query used to create the Recordset.

### *Specifying User Name/Password*

Done through 'Authentication' Tab. This tab is optional if security is not important. For Oracle and SQL Server, this is very important.

Having set these properties, we can use our Adodc. Add Textboxes and Labels. Make the Textboxes Data-bound. For details, refer to the section 'Binding TextBoxes to Data Control' in the DAO Data Control.

### *Common Properties of both Data Controls*

Both the data controls - DAO and ADO - share the following features. For details, refer to the corresponding head under DAO Data Control:

- Navigating through the Recordset
- Editing records
- Adding records
- Deleting records
- Retrieving a count of records

### *Searching for Specific Information*

This feature is somewhat different from the DAO Data Control. The difference is in specifying the commandtype for Adodc. This parameter is not available in DAO. In Adodc, this parameter is very important, because the command object is at the core of the search process in Adodc.

To begin the search process:

1. Start a new Standard EXE Project.
2. Add an ADO Data Control to the form.
3. Add the required number of Textboxes and Labels.
4. Add a command button cmdsearch for starting the search operation.
5. Code for cmdsearch button as follows:

Code Listing ado.1

```
PRIVATE SUB cmdsearch_CLICK( )
1. DIM datjoin AS DATE
2. DIM strsql AS STRING
3. datjoin = INPUTBOX("Which joining date to search for..?")
4. strsql = "SELECT*FROM employee WHERE date_join = #" & datjoin & "#"
5. adodc1.COMMANDTYPE = ADCMDTEXT       'this line is new in ADODC
6. adodc1.RECORDSOURCE = strsql
7. adodc1.REFRESH
  .END SUB
```

In the code above, line no. 5 is the only new line compared to the corresponding DAO Data Control search. For a complete explanation of the code, refer to searching options for DAO Data Control.

The remaining aspects are the same as those for the DAO Data Control:

- No special characters for numbers.
- Single quote [ ' ] for strings.
- Hash symbol [ # ] for dates. (For Access, else use single quote)

For more information on search-related issues, refer to the corresponding section in 'DAO no-code approach to databases'.

***Filtering Records***

ADO search has another option - the filter property. The filter, true to its name, filters out the unwanted records and only shows the records matching our criteria in the recordset. Code for the search on a button named cmdsearch:

Code Listing ado.2

```
PRIVATE SUB cmdsearch_CLICK( )
DIM int_srch AS INTEGER
1.  int_srch = INPUTBOX("Enter the salary to search")
2.  adodc1.RECORDSET.FILTER = "sal = " & int_srch
3.  IF adodc1.RECORDSET.EOF THEN MSGBOX "No data found"
4.  END SUB
```

This code works as follows:

- Line no. 1 is self-explanatory.
- Line no. 2 sets the filter property to a value given by the user. Records not matching the criteria are made 'invisible'.
- Line no. 3 checks for the EOF stage. By default, the recordset pointer will be at the first record after applying the filter. If no match is found, the pointer will be at EOF. Thus, by checking the EOF, we come to know whether matching data was found or not.

The filter criteria's syntax is similar to where clause of SQL. If no match is found, the bound textboxes will become blank.

***Advanced properties of ADO Data Control***

The following are the important properties, which will be covered in the section 'Code-based approach to ADO' that shortly follows.

- CursorLocation
- Cursortype
- LockType

# 8.7 MONITORING CHANGES TO THE DATABASE

To start using ADO create a new Standard EXE project in your VB6 environment. You'll need to add a reference to the ADO library using the Project -> References menu and selecting the "Microsoft ActiveX Data Objects 2.5 Library" or the "Microsoft ActiveX Data Objects 2.6 Library". Then on the form add a single command button and add the code below to the click event of the button.

Some data processing with ADO control is described below:

## 8.7.1 Connecting to a Database

With an Access database it is possible to connect to the database in 2 ways, JET or ODBC. The ODBC file management is easier; to change the filename or path just use the ODBC administrator in the control panel.

Dim ad as ADODB.Connection

```
set ad=New ADODB.Connection
Let ad.ConnectionString= "ODBC;DSN=" & DatabaseName & ";UID=" &
UserName & ";PWD=" & UserPassword
ad.Open
```

## 8.7.2 Opening a Table/Query for Viewing

Now we have the database connection established it is time to look at the data. The following example show how to open a table/query and move through it.

```
dim ar as ADODB.recordset
set ar=new adodb.recordset
ar.open {SQL Statement}
do while not ar.EOF
```

'Put the code here for what to do with the information.

'The field information can be access by the field name

```
intID=ar!IDField
'Or by the order number it is in the list (starting at 0)
intString=ar.Field(1).value
ar.movenext
loop
```

## 8.7.3 Change a Record

To edit/add/delete a record we can do it either using SQL or directly. Both DAO and ADO use the execute method for doing updates by SQL.

```
dim ar as ADODB.recordset
set ar=new adodb.recordset
```

```
ar.open {SQL Statement}
ar.execute "INSERT INTO tb(ID,Name) VALUES (10,Anne)"
```

These examples add a new record to the database directly.

Add new record

```
dim ar as ADODB.recordset
set ar=new adodb.recordset
ar.open {SQL Statement}
ar.addnew
ar!ID=intID
ar!Name=strName
ar.update
```

These examples show how to edit a record directly, after the recordset is open it checks that there is a record meeting the criteria in the open SQL. If not it creates one.

Edit Record

```
dim ar as ADODB.recordset
set ar=new adodb.recordset
ar.open "SELECT * FROM Tb WHERE tdID=10"
if ar.eof then
ar.addnew
else
ar.edit
end if
ar!ID=intID
ar!Name=strName
ar.update
```

These examples show how to delete a record directly, after the record set is open it checks that there is a record meeting the criteria in the open SQL. If not it does not do a delete.

ADO - Delete Record

```
Dim ar as ADODB.recordset
set ar=new adodb.recordset
ar.open "SELECT * FROM Tb WHERE tdID=10"
if not ar.eof then
ar.delete
end if
```

*Note:* If you open an object when you have finished with it, close it and set it to nothing. For example...

```
rs.close
set rs=nothing
```

This is good programming practice and clears the memory.

## 8.8 SQL BASIC

SQL stands for Structured Query Language and it is an American National Standards Institute (ANSI) language. It is used to access and manipulate data within relational databases such as MS Access, Oracle, DB2, and Sybase, to name a few. In order to conform to the ANSI standard they must support the same major keywords (SELECT, UPDATE, DELETE, etc.) in a consistent manner.

*Flavors of SQL*

SQL comes in many flavors. Oracle databases utilize their proprietary PL/SQL. Microsoft SQL Server makes use of Transact-SQL. However, all of these variations are based upon the industry standard ANSI SQL. We'll stick to ANSI-compliant SQL commands that will work on any modern relational database system.

*DDL and DML*

SQL commands can be divided into two main sublanguages. The Data Definition Language (DDL) contains the commands used to create and destroy databases and database objects. After the database structure is defined with DDL, database administrators and users can use the Data Manipulation Language to insert, retrieve and modify the data contained within it.

*Data Definition Language:* The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

CREATE

Installing a Database Management System (DBMS) on a computer allows you to create and manage many independent databases. For example, you may want to maintain a database of customer contacts for your sales department and a personnel database for your HR department. The CREATE command can be used to establish each of these databases on your platform. For example, the command:

CREATE DATABASE employees creates an empty database named "employees" on your DBMS. After creating the database, our next step is to create tables that will contain data. Another variant of the CREATE command can be used for this purpose. The command:

CREATE TABLE personal_info (first_name char(20) not null, last_name char(20) not null, employee_id int not null) establishes a table titled "personal_info" in the current database. In our example, the table contains three attributes: first_name, last_name and employee_id.

USE

The USE command allows you to specify the database you wish to work with within your DBMS. For example, if we're currently working in the sales database and want to issue some commands that will affect the employee's database, we would preface them with the following SQL command:

USE employees

It's important to always be conscious of the database you are working in before issuing SQL commands that manipulate data.

**ALTER**

Once you've created a table within a database, you may wish to modify the definition of it. The ALTER command allows you to make changes to the structure of a table without deleting and recreating it. Take a look at the following command:

ALTER TABLE personal_info

ADD salary money null

This example adds a new attribute to the personal_info table - an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

**DROP**

The final command of the Data Definition Language, DROP, allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

DROP TABLE personal_info

Similarly, the command below would be used to remove the entire employee's database:

DROP DATABASE employees

Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

***Data Manipulation Language (DML):*** The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

**INSERT**

The INSERT command in SQL is used to add records to an existing table. Returning to the personal_info example from the previous section, let's imagine that our HR department needs to add a new employee to their database. They could use a command similar to the one shown below:

INSERT INTO personal_info

values('bart','simpson',12345,$45000)

Note that there are four values specified for the record. These correspond to the table attributes in the order they were defined: first_name, last_name, employee_id, and salary.

**SELECT**

The SELECT command is the most commonly used command in SQL. It allows database users to retrieve the specific information they desire from an operational database. Let's take a look at a few examples, again using the personal_info table from our employee's database.

The command shown below retrieves all of the information contained within the personal_info table. Note that the asterisk is used as a wildcard in SQL. This literally means "Select everything from the personal_info table."

```
SELECT *

FROM personal_info
```

Alternatively, users may want to limit the attributes that are retrieved from the database. For example, the Human Resources department may require a list of the last names of all employees in the company. The following SQL command would retrieve only that information:

```
SELECT last_name

FROM personal_info
```

Finally, the WHERE clause can be used to limit the records that are retrieved to those that meet specified criteria. The CEO might be interested in reviewing the personnel records of all highly paid employees. The following command retrieves all of the data contained within personal_info for records that have a salary value greater than $50,000:

```
SELECT *

FROM personal_info

WHERE salary > $50000
```

**UPDATE**

The UPDATE command can be used to modify information contained within a table, either in bulk or individually. Each year, our company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:

```
UPDATE personal_info

SET salary = salary * 1.03
```

On the other hand, our new employee Bart Simpson has demonstrated performance above and beyond the call of duty. Management wishes to recognize his stellar accomplishments with a $5,000 raise. The WHERE clause could be used to single out Bart for this raise:

```
UPDATE personal_info

SET salary = salary + $5000

WHERE employee_id = 12345
```

**DELETE**

Finally, let's take a look at the DELETE command. You'll find that the syntax of this command is similar to that of the other DML commands. Unfortunately, our latest corporate earnings report

didn't quite meet expectations and poor Bart has been laid off. The DELETE command with a WHERE clause can be used to remove his record from the personal_info table:

DELETE FROM personal_info

WHERE employee_id = 12345

***Join Statement***

It's time to move on to one of the most powerful concepts the language has to offer – the JOIN statement. Quite simply, these statements allow you to combine data in multiple tables to quickly and efficiently process large quantities of data. These statements are where the true power of a database resides.

First, let us see the use of a basic JOIN operation to combine data from two tables. In later we'll explore the use of outer and inner joins to achieve added power.

We'll continue with our example using the PERSONAL_INFO table, but first we'll need to add an additional table to the mix. Let's assume we have a table called DISCIPLINARY_ACTION that was created with the following statement:

CREATE TABLE disciplinary_action (action_id int not null, employee_id int not null, comments char(500))

This table contains the results of disciplinary actions on company employees. You'll notice that it doesn't contain any information about the employee other than the employee number. It's then easy to imagine many scenarios where we might want to combine information from the DISCIPLINARY_ACTION and PERSONAL_INFO tables.

Assume we've been tasked with creating a report that lists the disciplinary actions taken against all employees with a salary greater than $40,000. The use of a JOIN operation in this case is quite straightforward. We can retrieve this information using the following command:

SELECTpersonal_info.first_name,personal_info.last_name,disciplinary_action.comments

FROM personal_info, disciplinary_action

WHERE personal_info.employee_id = disciplinary_action.employee_id

AND personal_info.salary > 40000

As you can see, we simply specified the two tables that we wished to join in the FROM clause and then included a statement in the WHERE clause to limit the results to records that had matching employee IDs and met our criteria of a salary greater than $40,000.

## 8.9 DATA OBJECTS

For working with the database object, we take the following steps:

1. 'declare' a variable of the type database.
2. 'create' an object of type database (object variable creation).

Once the 'object variable' is created, we use it to work with the database object programmatically. To begin working with it now, we start a new Standard EXE Project and add the DAO object library reference to it. Once this is done, we come to the code window and type the following code in the general/declarations section:

PRIVATE db AS DATABASE

This private variable will be made available to the entire Form. Now we type this line of code in the Form_load event:

SET db = OPENDATABASE (complete path of database)

The set keyword is very important in VB and is used whenever we refer to an 'object variable' - database, in the present case. This keyword is used when we address a complex datatype on the right hand side. Currently, we are referring to the database object on the right hand side. Thus, we have used the set keyword when creating the 'object variable' db. An 'object variable' is a variable, but of a special type : referring to a complex datatype. This means that we don't use the set keyword for simple datatypes such as Date, String, Integer, etc.

## 8.10 ADO

Microsoft's latest set of data access objects are the ActiveX Data Objects (ADO). These objects let you access data in a database server through any OLE DB provider. ADO is intended to give you a consistent interface for working with a wide variety of data sources, from text files to ODBC relational databases to complex groups of databases.

The way Microsoft implements connections to all those data sources is with the OLE DB set of COM interfaces, but that standard is a very complex one. Our interface to that interface, so to speak, is ADO, a set of objects with properties, events, and methods. Here are the ADOs:

- *Connection:* Access from your application to a data source is through a connection, the environment necessary for exchanging data. The Connection object is used to specify a particular data provider and any parameters.
- *Command:* A command issued across an established connection manipulates the data source in some way. The Command object lets ADO make it easy to issue commands.
- *Parameter:* Commands can require parameters that can be set before you issue the command. For example, if you require a debit from a charge account, you would specify the amount of money to be debited as a parameter in a Parameter object.
- *Recordset:* If your command is a query that returns data as rows of information in a table, then those rows are placed in local storage in a Recordset object.
- *Field:* A row of a Recordset consists of one or more fields, which are stored in Field objects.
- *Events:* ADO uses the concept of events, just like other interface objects in Visual Basic. You use event handling procedures with events. There are two types of events: ConnectionEvents (issued when transactions occur, when commands are executed, and when connections start or end) and RecordsetEvents (events used to report the progress of data changes).

Visual Basic 6 obsoletes the previously used database access technology provided by Jet and provides a new one known as ADO or Active Data Objects. This technology allows users to access data easily

from many existing databases (such as Access or Paradox) or from ODBC compliant databases like Oracle or MS SQL Server.

Using ADO is quite simple and allows programmers to provide flexible database front ends to users that are reliable and include many features. As a VB6 programmer ADO is important to know as most commercial VB programming exercises involve providing a front end to some sort of database.

Following are some of the key objects found in the ADO object model and some of their key methods and properties.

## 8.10.1 Connection Object

This object represents an open connection to the data source. This connection can be a local connection (say App.Path) or can be across a network in a client server application. Some of the methods and properties of this object are not available depending on the type of data source connected to.

***Key Properties***

**Table 8.1**

| Name | Data Type | Description |
|---|---|---|
| ConnectionString | String | Defines in string form the location of the data source you wish to connect to. Key fields in the string you will use are the "Provider=" and the "Data Source=" fields. You may also use the "Mode=" field. See descriptions of those properties for a more detailed view of each one. Some examples of connection strings follow:<br>**Data Source=c:\test.mdb;Mode=Read\|Write;Persist** - Connects to an Access database with read/write/persist permissions<br>**driver={SQL Server)**<br>**server=bigsmile;uid=sa;pwd=pwd;database=pubs** - Connects to an SQL Server database called bigsmile as user save with password pwd to database pubs. |
| Provider | String | A string defining the provider of a connection object. An example follows:<br>**Provider=Microsoft.Jet.OLEDB.4.0** -This string connects to a MS Jet 4.0 compliant database (an Access DB for example.) |
| Mode | ConnectModeEnum | Sets (or returns) the permissions for modifying data across the open connection. Available permissions include Read, Write, Read/Write, and various Share/Deny types. |
| CursorLocation | CursorLocationEnum | Sets the location of the cursor engine. You can select client side or server side, for simpler databases (like Access) client side is the only choice. |
| ConnectionTimeout | Long | Time in seconds that a connection will attempt to be opened before an error is generated. |
| CommandTimeout | Long | Time in seconds that an command will attempt to be executed before an error is generated. |

*Key Methods*

Table 8.2

| Name | Description |
|---|---|
| Close | Closes an open connection. |
| Open | Opens a connection with the settings in the ConnectionString property. |

## 8.10.2 Command Object

A command object specifies a specific method you intend to execute on or against the data source accessed by an open connection.

*Key Properties*

Table 8.3

| Name | Data Type | Description |
|---|---|---|
| ActiveConnection | conConnection as ADODB.Connection | Defines the Connection object the command belongs to. |
| CommandText | String | Contains the text of the command you want to execute against a data source. This can be a table name, a valid SQL string, or the name of a stored procedure in the data source. Which one you use is determined by the CommandType property. |
| CommandType | CommandTypeEnum | Defines the type of the command. The three most commonly used would be adCmdText, adCmdTable, and adCmdStoredProc. The setting adCmdText causes the CommandText string to be evaluated as an SQL string. The setting adCmdTable causes the CommandText string to be evaluated as a table name and will return the all of the records and columns in a table. The setting adCmdStoredProc causes the CommandText string to be evaluated as a stored procedure in the data source. |

*Key Methods*

Table 8.4

| Name | Description |
|---|---|
| Execute | Executes the query, SQL statement, or stored procedure specified in the CommandText property and returns a RecordSet object. |

## 8.10.3 RecordSet Object

The RecordSet object represents a complete set of records from an executed command or from an underlying base table in the database. A key thing to note is that a RecordSet object references only one record at a time as the current record.

## *Key Properties*

**Table 8.5**

| Name | Data Type | Description |
|---|---|---|
| CursorLocation | CursorLocationEnum | This sets or returns the location of the cursor engine for the database. The options are client side and server side, for less advanced databases like Access you may only be able to select client side. |
| CursorType | CursorTypeEnum | Sets the cursor type. CursorType typically determines when and to whom database changes are immediately visible to. For client side cursor locations only one type of CursorType is available, adOpenStatic. |
| EOF and BOF | Boolean | End Of File and Beginning Of File. When at the first record of an open RecordSet BOF will be true, when at the last EOF will be true. If you ever return a RecordSet with no records then EOF and BOF will be true. This provides an ideal way of testing if any records have been returned by a query. |
| Fields | Collection | Returns a collection of the field objects for an open record set. Database fields can be accessed by their name using the RecordSet!FieldName schema, by their index RecordSet.Fields(intIndex) or sometimes by their name from the fields collection RecordSet.Fields("FieldName"). I find in the situation where you know a database structure that the RecordSet!FieldName method is best, where structure is not known, then the RecordSet.Fields(intIndex) may be best. |
| LockType | LockTypeEnum | sets the lock type on records when they are open for editing. If a client side cursor location is selected then only optimistic and batch optimistic locking are available. |
| RecordCount | Long | Returns the number of records in an open RecordSet. If for some reason ADO cannot determine the number of records then this will be set to -1. |

## *Key Methods*

**Table 8.6**

| Name | Description |
|---|---|
| AddNew | Sets up an open record set to add a new record, once the required values have been set call the Update (or UpdateBatch) method to commit the changes. |
| Close | Closes an open RecordSet object, make sure that any changes are committed using the Update (or UpdateBatch) method before closing or an error will be generated. |
| MoveNext | Causes an open RecordSet object to move to the next record in the collection, if the current record is the last record then EOF is set to true. |
| MoveFirst | Causes an open RecordSet object to move to the first record in the collection. |
| MoveLast | Causes an open RecordSet object to move to the last record in the collection. |
| Open | Opens the RecordSet, typically this is done with a valid Command object that includes the command text (or SQL) to get the records you want. |
| Update | Causes any changes made to the current record to be written to disk. |
| UpdateBatch | Causes any changes you have made in batch mode to be written to disk. The way to use batch updates is to open the RecordSet object with the LockType adLockBatchOptimistic. |

### 8.10.4 ADO using Process

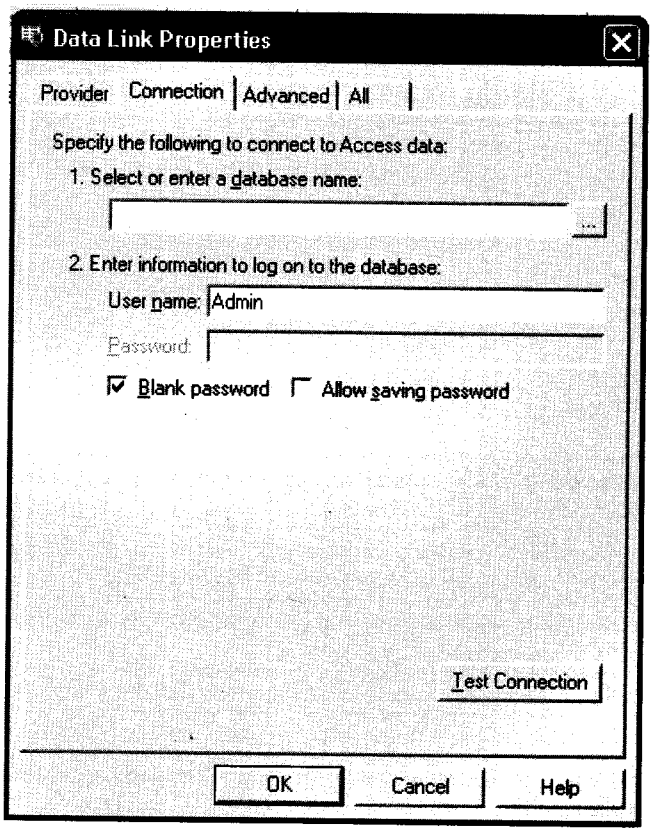
ADO using is a three steps process:

1. Define and open a Connection to a data source
2. Decide what data you need from the data source and define Command objects that will retrieve this data.
3. Retrieve the data you require by executing the command objects and manipulate the data using RecordSet objects.

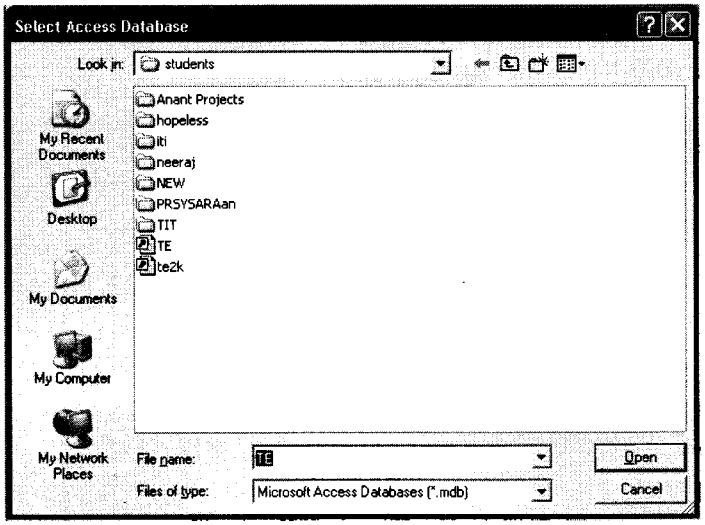
## 8.11 OLE.DB

*OLE DB for Access*

1. Select 'Microsoft Jet 4.0 OLE DB Provider' from the OLE DB providers dialog box. The 'Data Link Properties' dialog box for



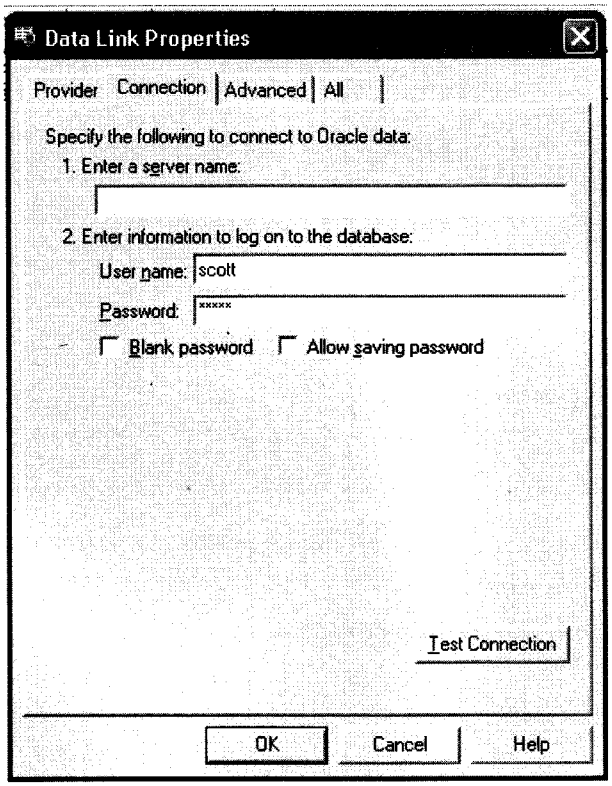

2. On the dialog box, click on the button with an ellipsis. The 'Select Access Database' dialog box appears.




3. Select the desired database and click on 'Open'. Dialog box closes.
4. If required, enter a valid 'User name' and 'Password'.
5. As always, click on 'Test Connection' to check the connectivity with Access.

*OLE DB for Oracle*

1. Select 'Microsoft OLE DB Provider for Oracle' from the OLE DB providers dialog box. The 'Data Link Properties' dialog box for Oracle opens up.

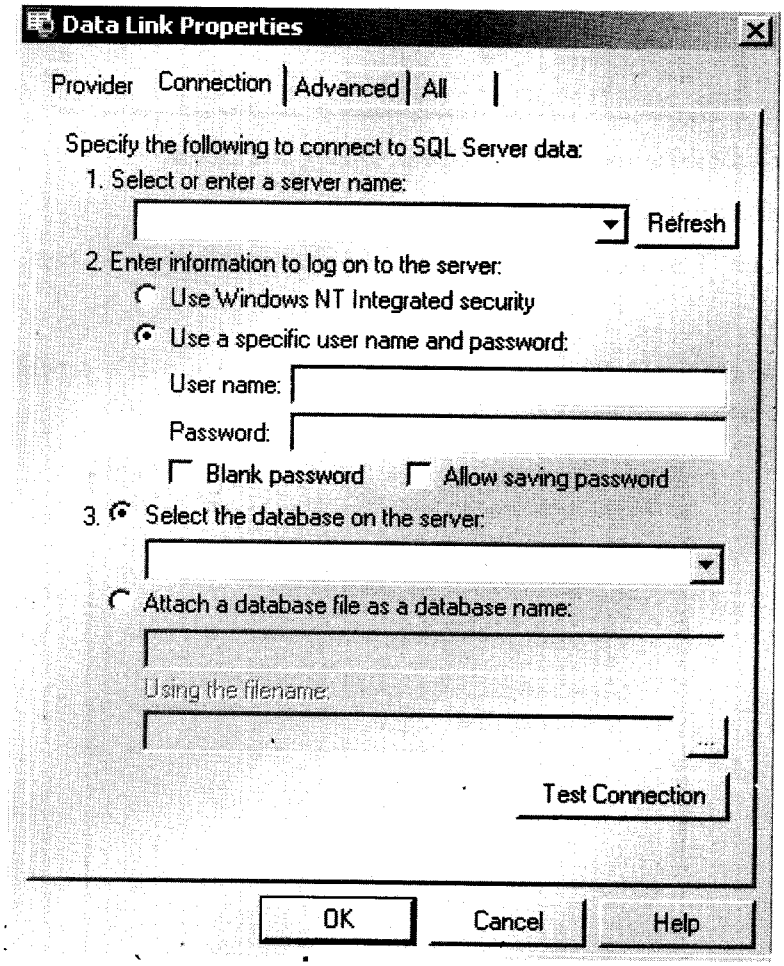


2. Leave the 'Server Name' blank.
3. Enter a valid 'User name' and 'Password'.
4. As always, click on 'Test Connection' to check the connectivity with Oracle.

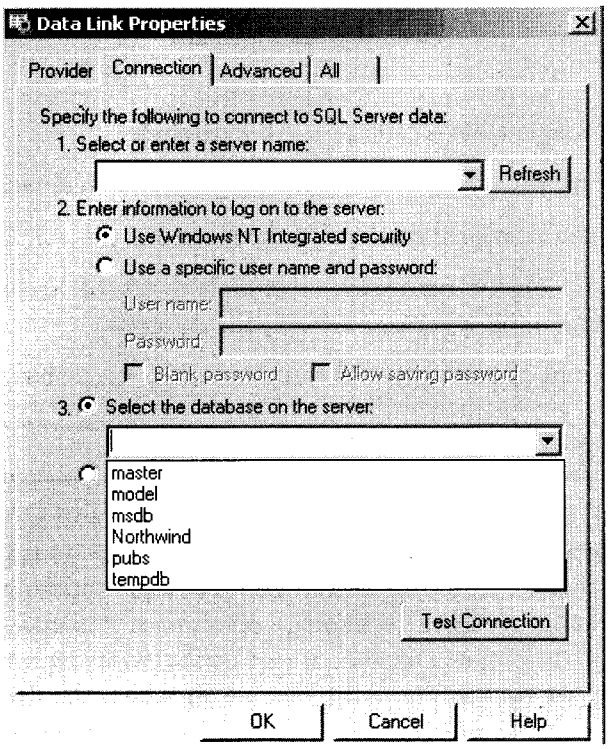
Oracle and SQL Server services should be running when connectivity is tried.

*OLE DB for SQL Server*

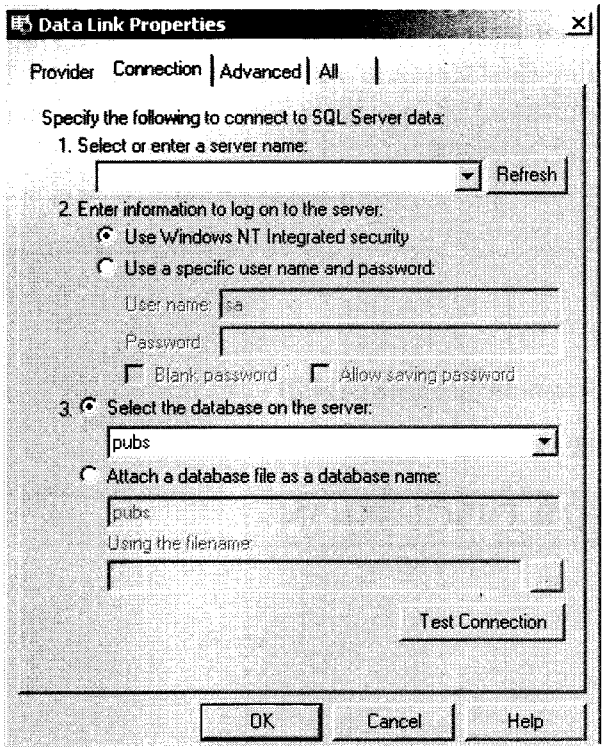
1. Select 'Microsoft OLE DB Provider for SQL Server' from the OLE DB providers dialog box. The 'Data Link Properties' dialog box for SQL Server opens up.




2. Select the security option (this depends on how security has been configured during installation - Windows NT integrated security or SQL Server user name/password)
3. Select the database name from the drop-down list.




4. Choose a database name on the server.
5. Specify a database file to attach.




6. Click on 'Test Connection' to check the connectivity and see the results.

**Check Your Progress**

Fill in the blanks:

1. A database using a single table is called a ..... database.
2. A database based on the ..... is known as a *relational database.*
3. We organize data in a relational database in the form of.....
4. The ..... object represents a complete set of records from an executed command or from an underlying base table in the database.
5. ADO uses the concept of events, just like other ..... objects in Visual Basic.

## 8.12 LET US SUM UP

Databases are designed to offer an organized mechanism for storing, managing and retrieving information. They do so through the use of tables. If you're familiar with spreadsheets like Microsoft Excel. One of the most important aspects of working in VB is programming for databases. When we say the world is witnessing an information technology revolution, it is important to understand that it is data and databases that are driving it. Database is one of the most important resources of any good company. Most of the time, we do not know the manner in which the data is maintained internally. For the end-user, the data is always presented in a simple, easy to understand format. With the Data Control, the implementation details of database-related tasks are hidden from us. The Data Control is based on the DAO model. It offers a simple, easy-to-use interface. The ADO Data Control (Adodc) is not available in the toolbox by default. To get it into the toolbox, we follow the by-now familiar route of clicking on 'Project' 'Components'. To start using ADO create a new Standard EXE project in your VB6 environment. You'll need to add a reference to the ADO library using the Project -> References menu and selecting the "Microsoft ActiveX Data Objects 2.5 Library" or the "Microsoft ActiveX Data Objects 2.6 Library". SQL stands for Structured Query Language and it is an American National Standards Institute (ANSI) language.

## 8.13 KEYWORDS

*ADO:* ActiveX Data Object

*Adodc:* ADO Data Control

*ANSI:* American National Standards Institute (ANSI)

*SQL:* Structured Query Language

## 8.14 QUESTIONS FOR DISCUSSION

1. Explain the term:
   (a) Modern Database
   (b) Data Manager
   (c) Data Control
   (d) Database Objects

2. Explain the programming of ADO with data control.
3. How can you monitor changes in the database?
4. What is the difference between UPDATE and DELETE in SQL?
5. Explain the command object in ADO.
6. Discuss the use of OLE.DB for SQL server.

**Check Your Progress: Model Answers**

1. flat
2. relational model
3. tables
4. RecordSet
5. interface

## 8.15 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

# UNIT V

# LESSON

# 9

# REPORTS

CONTENTS



## 9.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of data reports
- Describe the significance of printing reports
- Identify and explain the writing of data reports

## 9.1 INTRODUCTION

The Data Report Designer creates banded hierarchical reports. These are the most common type of database report; you've probably seen many of them, with headings, subheadings, details, and summaries organized in a hierarchical manner.

## 9.2 DATA REPORT

A Data Report is similar to a VB form in that it has a visual designer and a code module. Using the visual designer, you can divide the report into two or more sections, each with its own headings. Each section can contain controls to display the report details.

The design of the details sections is simplified by drag-and-drop functionality. The available controls are distinct from VB controls but have similar functionality.

In particular, the Function control lets you easily perform calculations on field data (sum, average, minimum, and maximum) and display the results as the report is generated. Headers and footers can be defined for the report as a whole and for each page of the report.

At run-time, output options are quite impressive. For instance, the Print Preview mode shows what the printout of the report will look like. Printing is a simple matter of calling the PrintReport method. Export to a file is supported in both HTML and text formats.

The Data Report Designer is a very impressive tool. It's not suitable for every type of report, but when it fits your needs, it can save you a tremendous amount of time.

## 9.3 DATA REPORT DESIGNER

The Microsoft Data Report designer is a versatile data report generator that features the ability to created banded hierarchical reports. Used in conjunction with a data source such as the Data Environment designer, you can create reports from several different relational tables. In addition to creating printable reports, you can also export the report to HTML or text files.

## 9.4 FEATURES OF THE DATA REPORT DESIGNER

The Data Report designer has several features:

- ***Drag-and-Drop Functionality for Fields:*** Drag fields from the Microsoft Data Environment designer to the Data Report designer. When you do this, Visual Basic automatically creates a text box control on the data report and sets the DataMember and DataField properties of the dropped field. You can also drag a Command object from the Data Environment designer to the Data Report designer. In that case, for each of the fields contained by the Command object, a text box control will be created on the data report; the DataMember and DataField property for each text box will be set to the appropriate values.
- ***Toolbox Controls:*** The Data Report designer features its own set of controls. When a Data Report designer is added to a project, the controls are automatically created on a new Toolbox tab named DataReport. Most of the controls are functionally identical to Visual Basic intrinsic controls, and include a Label, Shape, Image, TextBox, and Line control. The sixth control, the Function control, automatically generates one of four kinds of information: Sum, Average, Minimum, or

Maximum. For more information about the Function control, see "Adding a Function Control to the Data Report."

- *Print Preview:* Preview the report by using the Show method. The data report is then generated and displayed in its own window.

  *Note:* A printer must be installed on the computer to show the report in print preview mode.

- *Print Reports:* Print a report programmatically by calling the PrintReport method. When the data report is in preview mode, users can also print by clicking the printer icon on the toolbar.

  *Note:* A printer must be installed on the computer to print a report.

- *File Export:* Export the data report information using the ExportReport method. Formats for export include HTML and text.

- *Export Templates:* You can create a collection of file templates to be used with the ExportReport method. This is useful for exporting reports in a variety of formats, each tailored to the report type.

- *Asynchonous Operation:* The DataReport object's PrintReport and ExportReport methods are asynchronous operations. Using the ProcessingTimeout event, you can monitor the state of these operations and cancel any that are taking too long.

## 9.5 COMPONENTS OF THE DATA REPORT

The Data Report designer consists of the following objects:

- *DataReport Object:* Similar to a Visual Basic form, the DataReport object has both a visual designer and a code module. Use the designer to create the layout of a report. You can also add code to the designer's code module to programmatically format controls or sections contained by the designer.

- *Section Object:* Each section of the Data Report designer is represented by a Section object in a Sections collection. At design time, each section is represented by a header that you can click to select the section, and the section's pane where you can place and position controls. Use the object and its properties to dynamically reconfigure a report before it is built.

- *Data Report Controls:* Special controls that only work on the Data Report designer are included with it. (Note: you cannot use Visual Basic's intrinsic controls, or any ActiveX controls, on the Data Report designer). These controls are found in the Visual Basic Toolbox, but they are placed on a separate tab named "DataReport."

## 9.6 SECTIONS OF THE DATA REPORT DESIGNER

The default Data Report designer contains these Sections:

- *Report Header:* This section contains the text that appears at the very beginning of a report, such as the report title, author, or database name. If you want the Report Header to be the first page in the report, set its ForcePageBreak property to rptPageBreakAfter.

- *Page Header:* This section contains information that goes at the top of every page, such as the report's title.

- *Group Header/Footer:* This section contains a "repeating" section of the data report. Each group header is matched with a group footer. The header and footer pair are associated with a single Command object in the Data Environment designer.
- *Details:* This section contains the innermost "repeating" part (the records) of the report. The details section is associated with the lowest-level Command object in a Data Environment hierarchy.
- *Page Footer:* This section contains the information that goes at the bottom of every page, such as the page number.
- *Report Footer:* This section contains the text that appears at the very end of the report, such as summary information, or an address or contact name. The Report Footer appears between the last Page Header and Page Footer.

## 9.7 DATA REPORT CONTROLS

When a new Data Report designer is added to a project, the following controls are automatically placed in the Toolbox tab named DataReport:

- *TextBox Control (RptTextBox):* It allows you to format text, or assign a DataFormat.
- *Label Control (RptLabel):* It allows you to place labels on the report to identify fields or sections.
- *Image Control (RptImage):* It enables you to place graphics on your report. Note that this control cannot be bound to a data field.
- *Line Control (RptLine):* It lets you draw rules on the report to further distinguish sections.
- *Shape Control (RptShape):* It enables you to place rectangles, triangles, or circles (and ovals) on a report.
- *Function Control (RptFunction):* It is a special text box that calculates values as the report is generated.

## 9.8 WRITING A SIMPLE DATA REPORT

This topic creates a simple data report using a Data Environment designer as a data source. The Data Environment designer uses the NorthWind database supplied with Visual Basic to create a simple hierarchical cursor. The cursor contains two tables, Customers and Orders, and uses the CustomerID field to link the two. The finished report resembles the Figure 9.1.

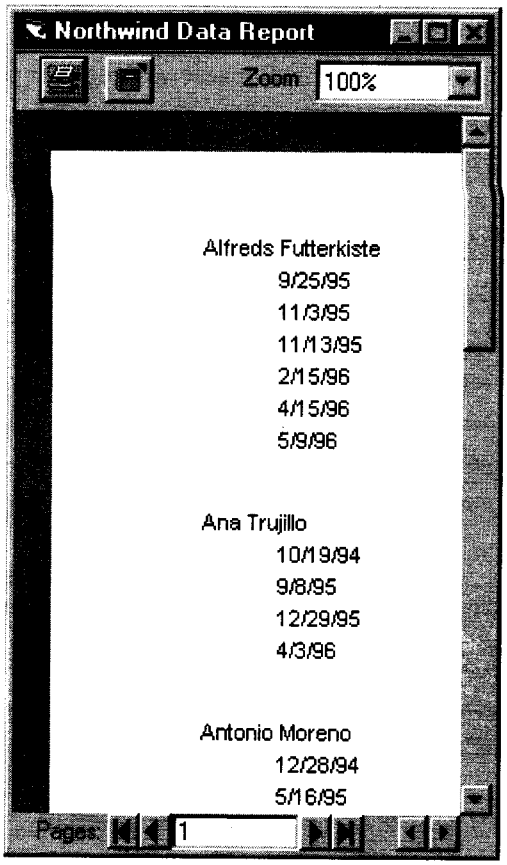



**Figure 9.1: Simple Data Report: Order Dates by Customers**

Before you begin the step-by-step process, ensure that the Northwind database (Nwind.mdb) is present on your computer. If it is not present, copy the file from your Visual Basic CD onto your hard disk.

### 9.8.1 To Create a Simple Hierarchical Cursor in the Data Environment Designer

1. Create a new Standard EXE project.
2. On the Project menu, click Add Data Environment to add a designer to your project. If the designer is not listed on the Project menu, click Components. Click the Designers tab, and click Data Environment to add the designer to the menu.

   *Note:* The first four kinds of ActiveX designers loaded for a project are listed on the Project menu. If more than four designers are loaded, the later ones will be available from the More ActiveX Designers submenu on the Project menu.
3. On the Data Link Properties dialog box, click Microsoft Jet 3.51 OLE DB Provider. This selects the correct OLE DB provider for accessing a Jet database.
4. Click the Next button to get to the Connection tab.
5. Click the ellipsis button (...) next to the first text box.
6. Use the Select Access Database dialog box to navigate to the nwind.mdb file, which is installed in the Program Files\Microsoft Visual Studio\Vb98 directory.

7. Click OK to close the dialog box.
8. Right-click the Connection1 icon, and click Rename. Change the name of the icon to Northwind.
9. Right-click the Northwind icon, and then click Add Command to display the Command1 dialog box. In the dialog box, set the properties as shown below:

| Property | Setting |
|---|---|
| Command Name | Customers |
| Connection | Northwind |
| DataBase Object | Table |
| Object Name | Customers |

10. Click OK to close the dialog box.
11. Right-click the Customers command, and click Add Child Command to display the Command2 dialog box. In the dialog box, set the properties as shown below:

| Property | Setting |
|---|---|
| Command Name | Orders |
| Connection | Northwind |
| DataBase Object | Table |
| Object Name | Orders |

12. Click the Relation tab. The Relate to a Parent Command Object check box should be checked. The Parent box should contain Customers; both the Parent Fields and Child Fields/Parameters boxes should contain CustomerID.
13. When designing relational databases, it's customary for related tables to use the same name for linking fields. In this case, the linking fields are both named CustomerID. The Data Environment designer automatically matches such pairs in the dialog box.
14. Click Add. Click OK to close the dialog box.

    Clicking the Add button adds the relation to the Command object. After closing the dialog box, the Data Environment designer reflects the relationship by displaying the two commands as a hierarchy. This hierarchy will be used to create the data report.
15. Set the properties of the project and designer according to the settings below, then save the project:

| Object | Property | Setting |
|---|---|---|
| Project | Name | prjNwind |
| DataEnvironment | Name | deNwind |
| Form | Name | frmShowReport |

### 9.8.2 Creating the Data Report

Once the Data Environment designer has been created, you can create a data report. Because not all of the fields in the data environment will be useful in a report, this series of topics creates a limited report that displays only a few fields.

*To create a new Data Report*

1. On the Project menu, click Add Data Report, and Visual Basic will add it to your project. If the designer is not on the Project menu, click Components. Click the Designers tab, and click Data Report to add the designer to the menu.

   *Note:* The first four kinds of ActiveX designers loaded for a project are listed on the Project menu. If more than four designers are loaded, the later ones will be available from the More ActiveX Designers submenu on the Project menu.

2. Set the properties of the DataReport object according to the table below:

| Property | Setting |
|---|---|
| Name | rptNwind |
| Caption | Northwind Data Report |

3. On the Properties window, click DataSource and then click deNwind. Then click DataMember and click Customers.

   Important To set the DataSource property to deNwind, the Data Environment designer must be open. If it is closed, press CTRL+R to display the Project window, then double-click the data environment icon.

4. Right-click the Data Report designer, and click Retrieve Structure.

   You have added a new group section to the designer. Each group section has a one-to-one correspondence to a Command object in the data environment; in this case, the new Group section corresponds to the Customers Command object. Notice also that the Group Header has a matching Group Footer section.

   *Note:* The Data Environment allows you to create hierarchies of Command objects wherein a Command object has more than one child object — child Command objects parallel to each other. The Data Report designer, however, is not as flexible, and can't display more than one child object at a time. In such cases, when executing a Retrieve Structure command, the Data Report will display only the first of the child commands, and none below it. Thus you should avoid creating Command hierarchies with parallel children commands.

5. From the Data Environment designer, drag the CompanyName field (under the Customers command) onto the Group Header (Customers_Header) section.

   The Group Header section can contain any field from the Customers command, however, for demonstration purposes, only the Customer name is displayed at this time.

6. Delete the Label control (rptLabel) named Label1.

If you do not want a Label control to be included with the TextBox control, you can uncheck the Drag and Drop Fields Caption option on the Field Mapping tab of the Data Environment designer's Options dialog box.

7. From the Data Environment designer, drag the OrderDate field (under the Orders command) onto the Details (Orders_Detail) section. Delete the Label control.

   The Details section represents the innermost "repeating" section, and thus corresponds to the lowest Command object in the Data Environment hierarchy: the Orders Command object.

8. Resize the Data Report designer's sections to resemble the figure below:

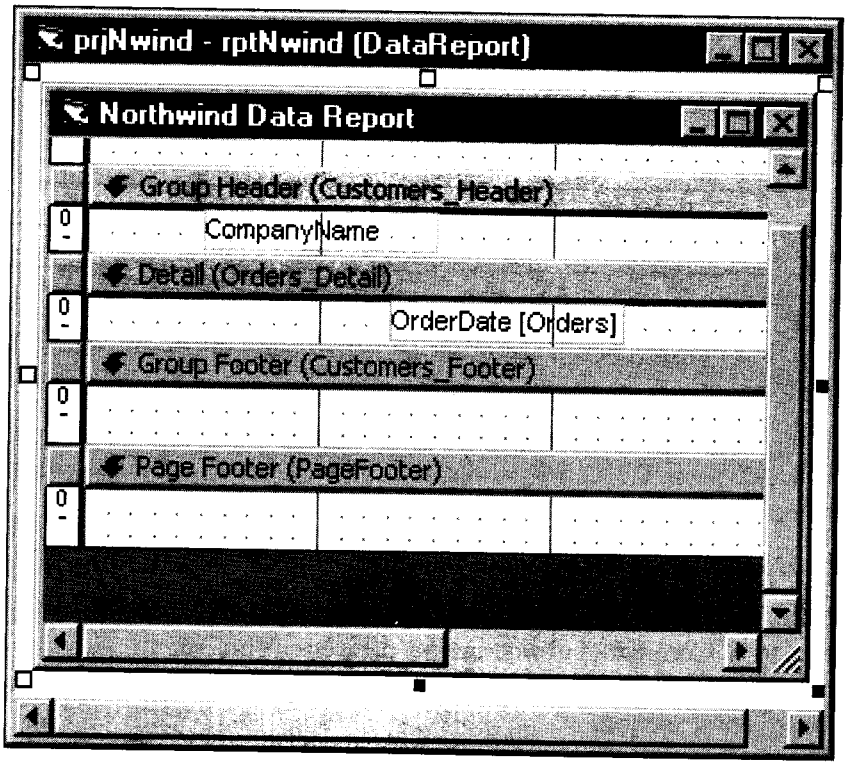



**Figure 9.2: Data Report Designer's Sections**

It's important to resize the height of the Details section to be as short as possible because the height will be multiplied for every OrderDate returned for the CompanyName. Any extra space below or above the OrderDate text box will result in unneeded space in the final report.

9. Save the project.

### 9.8.3 Preview the Data Report using the Show Method

Now that the data environment and the data report objects have been created, you are almost ready to run the project. One step remains: to write code to show the data report.

To show the data report at run time

1. On the Project Explorer window, double-click the frmShowReport icon to display the Form designer.

2. On Toolbox, click the General tab.
3. When you add a Data Report designer to your project, its controls are added to the tab named DataReport. To use the standard Visual Basic controls, you must switch to the General tab.

   Click the CommandButton icon and draw a CommandButton on the form.
4. Set the properties of the Command1 control according to the table below:

| Property | Setting |
|---|---|
| Name | cmdShow |
| Caption | Show Report |

5. In the button's Click event, paste the code below.

```
Private Sub cmdShow_Click()
rptNwind.Show
End Sub
```

6. Save and run the project.
7. Click Show Report to display the report in print preview mode.

### 9.8.4 Optional - Setting the Data Report as the Startup Object

You can also display the data report with no code at all.

1. On the Project menu, click prjNwind Properties.
2. In the Startup Object box, select rptNwind.
3. Save and run the project.

*Note:* If you use this method, you can remove the Form object from your project.

## 9.9 EXPORTING A DATA REPORT

After compiling a report you may wish to reuse it, either as part of a larger document or perhaps for distribution on an intranet or the Internet. The Data Report designer's ExportReport method allows you to accomplish these tasks. Using the ExportReport method, you can export any report as a text file or as an HTML file. Additionally, you can use any of a number of ExportFormat objects to tailor the content and appearance of an exported file.

The ExportReport method does not support the exporting of images or graphic shapes.

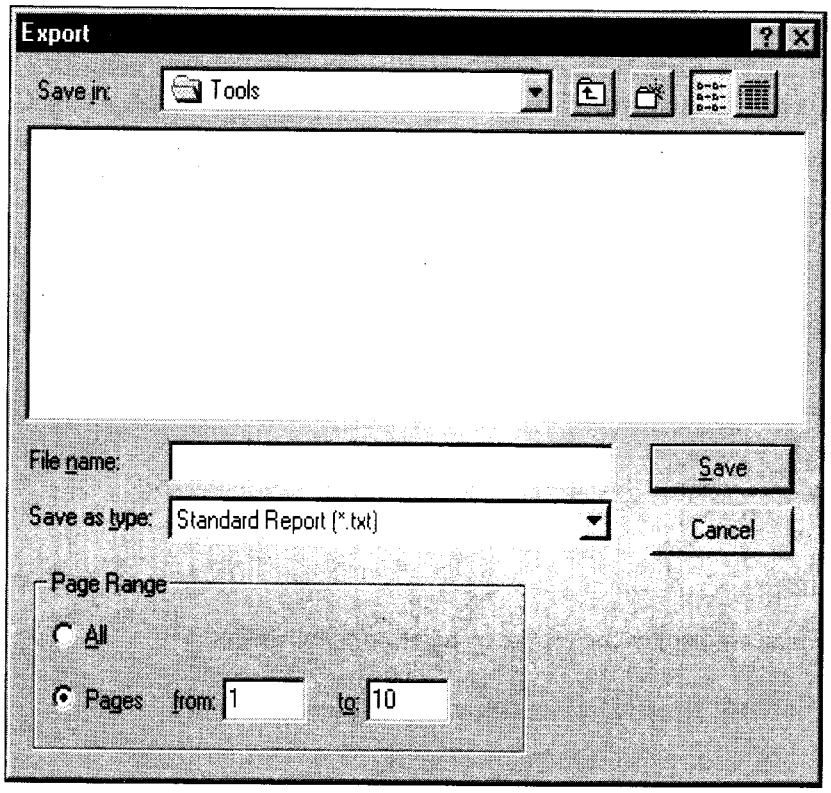



**Figure 9.3: Export Dialog Box**

## 9.10 PRINTING A DATA REPORT

Printing a data report can be accomplished in one of two ways. The user can click the Print button that appears on the data report in Print Preview mode (using the Show method), or you can programmatically enable printing using the PrintReport method. If an error occurs during printing, trap it in the Error event.

**Check Your Progress**

Fill in the blanks:

1. The Data Report Designer creates banded ..... reports.
2. When a Data Report designer is added to a project, the controls are automatically created on a new Toolbox tab named ..... .
3. Export the data report information is possible using the ..... method.
4. ..... enables you to place rectangles, triangles, or circles (and ovals) on a report.
5. ..... section of report contains information that goes at the top of every page, such as the report's title.
6. The ..... method does not support the exporting of images or graphic shapes.

## 9.11 LET US SUM UP

A Data Report is similar to a VB form in that it has a visual designer and a code module. Using the visual designer, you can divide the report into two or more sections, each with its own headings. Each section can contain controls to display the report details. The Microsoft Data Report designer is a versatile data report generator that features the ability to created banded hierarchical reports. After compiling a report you may wish to reuse it, either as part of a larger document or perhaps for distribution on an intranet or the Internet. The Data Report designer's ExportReport method allows you to accomplish these tasks but it does not support the exporting of images or graphic shapes.

## 9.12 KEYWORD

*Data Report Designer:* It is a versatile data report generator that features the ability to created banded hierarchical reports.

## 9.13 QUESTIONS FOR DISCUSSION

1. What are the similarities between data report and VB form?
2. Write short notes on:
   (a) Export Templates
   (b) Asynchonous Operation
3. Discuss features of the data report designer.
4. Describe data report exporting in Visual Basic.
5. What are the components of the data report?
6. What are default data report designer sections?
7. Compare and contrast between:
   - Page header and group header
   - Page footer and report footer
8. Write an essay on data report controls.
9. How to display the data report with no code?
10. How to print a data report?

**Check Your Progress: Model Answers**

1. Hierarchical
2. Data Report
3. Export Report
4. Shape control
5. Page Header
6. Export Report

## 9.14 SUGGESTED READINGS

*Visual Basic 6 Programming-Black Book*, Steven Holzner, Dreamtech Press Publisher, New Delhi.

*Programming Microsoft Visual Basic 6.0*, Francesco Balena, WP Publishers and Distributors.

*Visual Basic 6*, Gary Cronell, Tata McGraw Hill Publishing Company Ltd.

*Visual Basic 6 - How to Program*, H.M. Deitel., P.J. Deital and T.R.Nieto.

LESSON

# 10

# ERROR HANDLERS AND DEBUGGING TECHNIQUES

CONTENTS



## 10.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of error handling
- Describe the significance debugging techniques

## 10.1 INTRODUCTION

Having come this far, I hope we can - possibly - assume that we are getting a good hold on VB. Also that we are getting familiar with 'some of' the complexities involved. Programming is a fairly easy task in VB after all-with an interactive environment, intelli-sense and auto-complete, developing a program in VB is really easy.

However, when it comes to enterprise-level programming, the honeymoon with VB ends! The programs will give out a variety of errors, they will crash with a host of unwelcome messages - 'type mismatch', 'overflow', 'invalid use of null', etc. (I'm sure you can add some of your own statements to the list). Suddenly, you feel like banging your head against the wall ... But wait!! ... help is present nearby. How to handle these run-time devils - errors - will be the focus of this chapter.

## 10.2 ERROR-HANDLERS

Even if we have the best of brains, the best of quality control, the best of processes involved, we can never be sure that we will have a totally error-free program. Even a program as big as Windows XP has errors that are equally big in number. So much for software quality! Of course, this does not mean

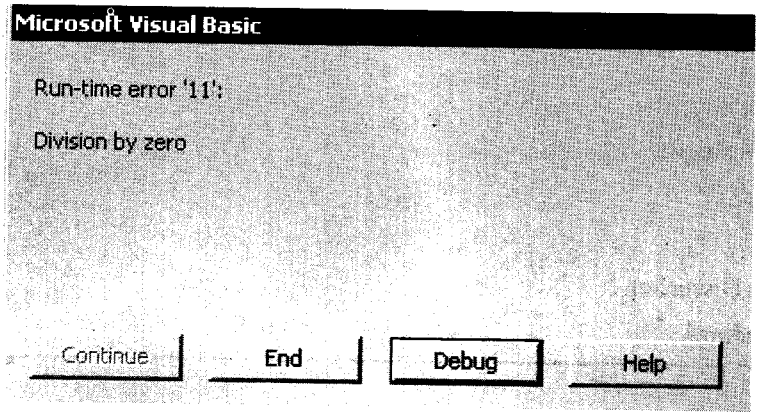
that the search for the error-free environment is over – the search is still continuing. Similar to other languages, VB also offers a variety of tools and options for handling and tackling run-time errors.

*Err Object and on Error Goto*

The err object is at the core of error-handling in VB. Coupled with the on error goto statement, it does most of the error-handling tasks for us. Consider the following code:

TEXT3 = VAL ( TEXT1 ) / VAL ( TEXT2 )

The code is quite simple and free from errors, as we can understand. However, if we assign '0' to 'text2', we get the following error message:




This is a pretty straightforward error message- we are trying to divide by 0. If we press on Help, the MSDN library will open up, giving details on the error that has occurred. If we press on Debug, the code window will open up, highlighting the line where the error occurred.

In a sense, our program has 'crashed'. To avoid such a situation, we will incorporate error-handling in our application, making use of Err object; this object is at the core of error-handling in VB6.

Besides others, the Err object exposes three very important properties:

| Property | Meaning |
|---|---|
| Number | The number of the error that was raised. Every trappable error in VB is associated with a number. |
| Description | A short description of the error that was raised. |
| Source | The library which raised the error-VB, ADO, etc |

With the help of Err, we code for the division (text1 / text2) as follows:

Code Listing ero.1

```
PRIVATE SUB command1_CLICK ( )
1.  ON ERROR GOTO problem
    'if an error occurs go to the area labelled as problem
2.  DIM intnum1 AS INTEGER
3.  DIM intnum2 AS INTEGER
4.  DIM div AS DOUBLE
```

```
5.  intnum1 = text1.TEXT
6.  intnum2 = text2.TEXT
7.  div = intnum1 / intnum2
8.  MSGBOX "DIVISION OVER"
9.  EXIT SUB
    'skip further lines, if no error encountered so far
10. problem:      'label for starting the error-handling routine
11. IF ERR.NUMBER = 11 THEN
12. MSGBOX ERR.NUMBER & " " & ERR.DESCRIPTION
    'error number will be printed on the screen, alongwith the description
13. END IF
    END SUB
```

This code works as follows:

- Line no. 1 informs VB that it should give up any error-handling on its own. Instead, it should go for error-handling as specified by us. The line also mentions that if an error occurs, it will be handled at a named area (label) called as problem.
- Line no. 9 makes the program come out of the procedure (sub) if no error has been encountered.
- Line no. 10 marks the beginning of the error-handling routine. This is the named area, which in VB we call as a 'label'. Once an error occurs in this program, all the lines following the error will be skipped over and the program control will shift to line no. 10, i.e., the label.
- All the lines following line no. 10 are executed, if an error is raised.

When we run the program now and give a value of '0' for division, the program will not raise an error and won't come in the break mode. Instead, it will display a messagebox informing the user on the error.

In order to temporarily disable the error-handling, we use the on error statement as follows:

ON ERROR GOTO 0

which will act as if no error-handling has been included in our application.

The Source property of Err object gives the name of the application which raised the error. For example, if we connect to MS-Word through VB code and it generates an error, the Err.Source will be set to Word.Application.

Another important property of Err is LastDllError. Errors raised when calling on Windows DLLs are not trappable in VB. This property gives a vital information for dealing with errors raised during Windows API calls.

### *Resume*

Once an error is raised, VB stops further program execution. With the resume keyword, VB will re-execute the line which caused the error. The right approach is to trap the error number and ask the user for a different input. This way, the program will 'resume' from the line where the problem has arisen. To incorporate resume, we modify our code as follows:

Code Listing ero.2

```
PRIVATE SUB command1_CLICK ( )
1.  ON ERROR GOTO problem
2.  DIM intnum1 AS INTEGER
3.  DIM intnum2 AS INTEGER
4.  DIM div AS DOUBLE
5.  intnum1 = text1.TEXT
6.  intnum2 = text2.TEXT
7.  div = intnum1 / intnum2
8.  MSGBOX "DIVISION OVER"
9.  EXIT SUB
10. problem:
11. IF ERR.NUMBER = 11 THEN ' if division by 0 error
12. Intnum2 = INPUTBOX ("Enter a New divisor") 'ask for a new divisor after
    error
13. RESUME ' re-execute the line that caused the error
14. ELSE
15. MSGBOX ERR.NUMBER & " " & ERR.DESCRIPTION
    'any other error number will be shown, alongwith the description
16. END IF
    END SUB
```

This code works as follows:

- Line no. 11 checks the error number. If it is 11 (i.e., division by Zero), line no. 12 is executed.
- Line no. 12 asks the user for another value of the variable 'intnum2'.
- Line no. 13 says 'Resume', which means the program control will shift to the line where the error was raised. However, the new value of 'intnum2' will be used in the calculation this time (as given in line no. 12).
- Line no. 15 will display the error number and description, if the error number is not 11.

*Resume Next*

As it suggests, Resume Next takes the program control to the line following the line where the error occurred. For instance, if error was raised on line no. 10, the program will continue from line no. 11. Resume Next is used if a particular error is worth ignoring and normal processing must continue even if that error has been raised.

If all the errors are worth ignoring, we can develop a more generic code by including the following line at the top of the procedure (sub):

ON ERROR RESUME NEXT

With this code, the program will never stop at any error- all errors generated in the program will be ignored. In fact, this program will not need any error-handling routine at all; the program will never stop at any error.

Though Resume Next appears very useful, it should be used with care. An ignored error can prove problematic at a later stage.

## 10.3 DEBUGGING TECHNIQUES

Error-handling is a vital aspect of any good program. Another equally interesting aspect is the debugging feature of the language.

Error handling is oriented towards the end-user. Debugging is oriented towards the developer of the program.

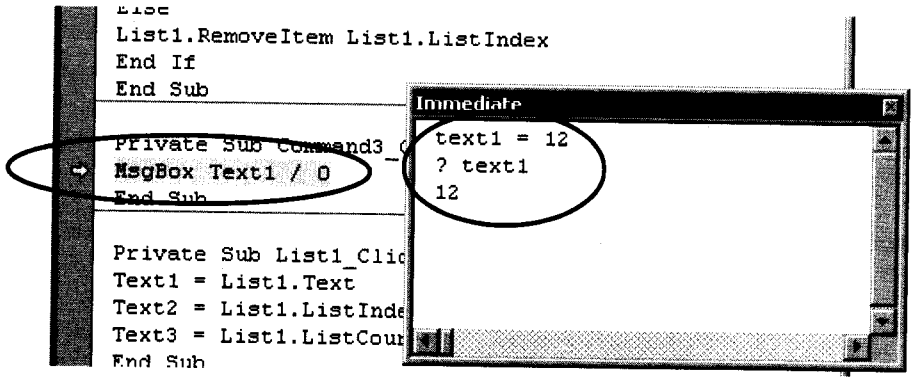
The following debugging options are available in VB6:

*Immediate Window*

This is a very elementary approach to debugging. It functions as follows:

1. This window pops up automatically whenever an error is encountered. At this stage, the program has entered the 'Break Mode'.
2. In the immediate window, type in a new value for the variable which caused the error.
3. Press enter. The new value will be assigned to the variable.
4. Press on start/continue. The program will resume from the 'Break Mode' and continue execution with the new value of the variable.

Here is how the immediate window appears. Also observe the code window, highlighting the line where the error was raised:




As per the figure, we have used the immediate window in two steps:

- Assigned a value to 'text1' on the 1st line and pressed 'Enter'.
- Queried the value of 'text1' on the next line.

We can also get the immediate window by pressing on 'Ctrl' + 'Pause Break'. This will bring up the window and bring the program in the 'Break Mode'.

***Debug Object***

The Debug object helps us in the debugging process (what else?) with two important methods:

Debug.Print: This method prints a value in the immediate window. If we write the code

DEBUG.PRINT intsum2

it will keep on printing the values of the variable intsum2 in the immediate window, whenever this line is executed. Even in a stopped state of the program, if we click on the menu bar at 'View' 'Immediate Window', the immediate window will show all the values of 'intsum2'. Thus, with debug.print we can trace how the values of variables are changing in our program.

Debug.Assert: Stops the program and highlights the line where the condition given by the user has gone wrong. If the condition is not fulfilled, VB assumes an error and brings the program in the break mode. For example, consider the line

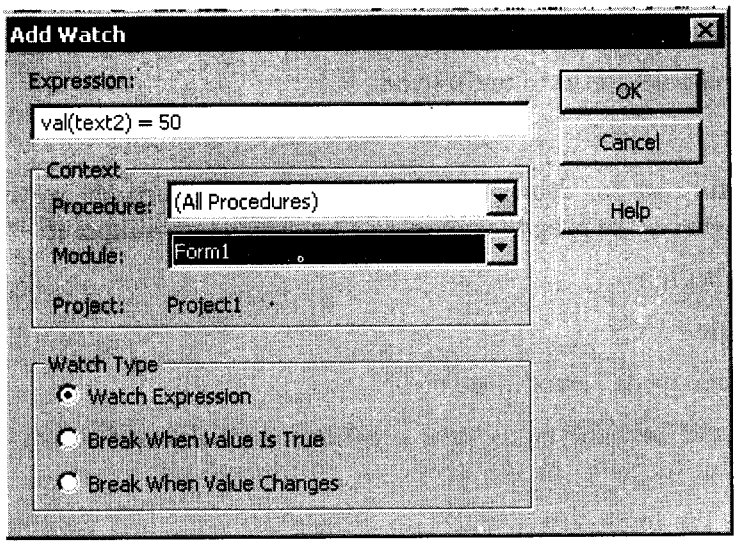
DEBUG.ASSERT VAL(text1) > 10

If the value of 'text1' is 20, the program will continue normally. If the value of 'text1' becomes less than '10', the program will automatically stop at this line and highlight this line. That is, debug.assert will temporarily stop program execution when the condition specified in debug.assert becomes False. With this approach, we can check exactly when is a value going wrong. Of course, once we come to know the error condition/value, we have to take appropriate measures to solve the problem.

If we forget to remove debug.print and debug.assert from our code when compiling, VB will comment out all of these lines.

### *Watch Window*

This 'window' helps to 'watch' the different values a variable is being assigned at different stages of the program. To get the window, click on 'Debug' 'Add Watch'. The following screen appears:

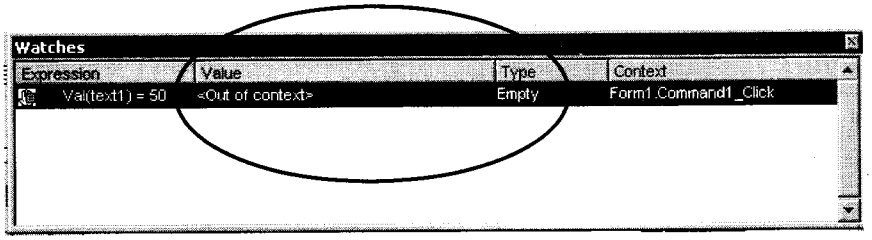



In the textbox at the top, specify any valid VB expression based on the variable to watch. Then we see two drop-down lists:

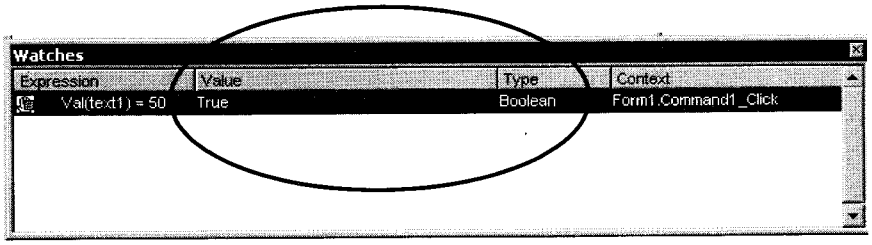
- In the 1st list, choose the procedure which will 'watch' this value or expression (Text1_Change, List1_Click, etc).
- In the 2nd list, choose the module where this value will be watched (Form1, Form2, and so on).

From the three option buttons towards the bottom, choose the most appropriate action to take. All of the options are self-explanatory.

Once we finish all of these steps, the watch window will appear as follows:




Once the watch condition becomes true at runtime, the watch window becomes 'active' and comes in the foreground, displaying the following information:



In the figure given above,

- The 'Value' column is mentioning True.
- The 'Type' column is mentioning Boolean.
- The 'Context' is specifying the module and procedure name where the value is being watched.

The meaning is clear enough for us to understand.

***Break Point***

A very easy and useful tool for debugging a program. In this approach, we assign a 'breakpoint' to some lines. Now, whenever the program comes to a line having a breakpoint:

- VB will stop at that line.
- Highlight that line.
- Come in the 'break mode', error or no error.
- In this state, we can take the mouse over a particular variable and see its value at that particular moment - the value 'pops up' on the screen.
- We can also use the immediate window to change the value of a variable and instantly see the changed result.

Setting a break point is very easy. Firstly, select the line where the point needs to be inserted. Now, we have three options available:

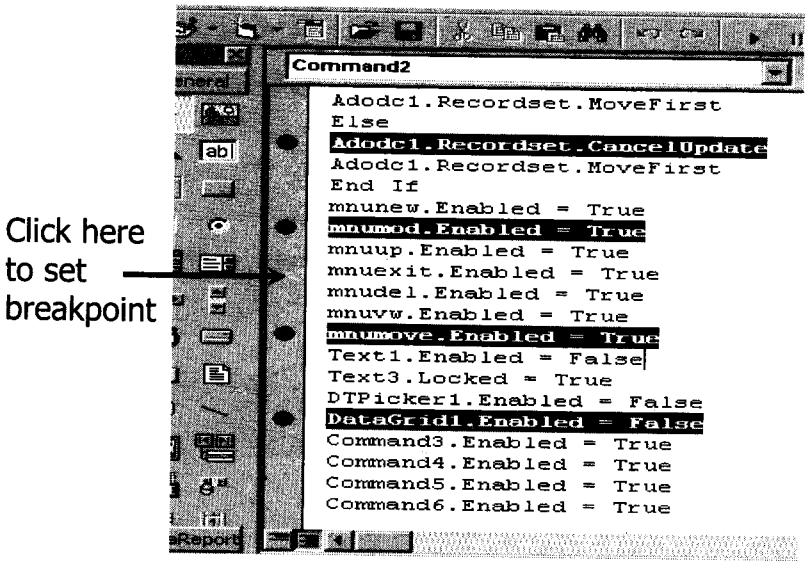
- On the menu bar, click on 'Debug' 'Toggle Breakpoint'.

  Or

- Press on F9 key.

  Or

- Click on the small grey area between the ToolBox and the code window, as shown here:



The lines with a break point are marked with a red spot, as the figure above shows. The red spots appear in the grey coloured area where we click to toggle a break point. Once the runtime check via the break point is over, we continue with further processing by pressing the start button or the F5 key.

*Step Into/Step Over/Step Out*

These options are concerned with how the program should continue, once it encounters a breakpoint and stops there. We come to the 'Debug' menu and we see the following options:

- *Step Into:* This is a single-step through the entire code. Any called procedures are executed line by line. Alternatively, we press F8.
- *Step Over:* This is also a single-step through the code, but any procedure calls are stepped over. Alternatively, we press Shift + F8.
- *Step Out:* The program steps out of the current procedure. Alternatively, we press 'Ctrl' + 'Shift' + F8.

Create simple tools for debugging through temporary TextBoxes, Labels, MessageBoxes, etc. These can display values of variables as we go through the normal execution, alongwith taking help of VB's built-in debugging tools.

**Check Your Progress**

Fill in the blanks:

1. The ..... object is at the core of error-handling in VB.
2. With the ..... keyword, VB will re-execute the line which caused the error.
3. This 'window' helps to 'watch' the different values a ..... is being assigned at different stages of the program.
4. This ..... pops up automatically whenever an error is encountered.

## 10.4 LET US SUM UP

Even if we have the best of brains, the best of quality control, the best of processes involved, we can never be sure that we will have a totally error-free program. Even a program as big as Windows XP has errors that are equally big in number. So much for software quality! Of course, this does not mean that the search for the error-free environment is over – the search is still continuing. Error-handling is a vital aspect of any good program. Another equally interesting aspect is the debugging feature of the language. Error handling is oriented towards the end-user. Debugging is oriented towards the developer of the program.

## 10.5 KEYWORDS

*Debugging:* Debugging is oriented towards the developer of the program.

*Resume Next:* Resume Next takes the program control to the line following the line where the error occurred.

*Debug Object:* The Debug object helps us in the debugging process.

## 10.6 QUESTIONS FOR DISCUSSION

1. What are the error handlers? What is the difference between the Resume and resume next?
2. Explain Err object.
3. Discuss the two main debugging techniques.
4. "The Debug object helps us in the debugging process with two important methods." Discuss.

**Check Your Progress: Model Answers**

1. err
2. resume
3. variable
4. window

## 10.7 SUGGESTED READINGS

*Visual Basic 6 Programming-Black Book*, Steven Holzner, Dreamtech Press Publisher, New Delhi.

*Programming Microsoft Visual Basic 6.0*, Francesco Balena, WP Publishers and Distributors.

*Visual Basic 6*, Gary Cronell, Tata McGraw Hill Publishing Company Ltd.

*Visual Basic 6 - How to Program*, H.M. Deitel., P.J. Deital and T.R.Nieto.

# LESSON

# 11

# DHTML

CONTENTS



## 11.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of internet
- Discuss how to identify the internet applications using visual basics

## 11.1 INTRODUCTION

Whether the Internet is considered a revolution or an evolution, it has changed human lives in a number of ways. It has entered practically every aspect of human life - education, news dissemination, science and technology, trade and commerce, and what not. Before going ahead, let us familiarize ourselves with some of the common terms used in the Internet domain.

The Internet is the world-wide public network of computers, sharing resources on a 24-hour 7-day basis. It is based on WAN technology and works through a common protocol, TCP/IP.

## 11.2 INTERNET

Internet is a gigantic computer network of many different computers. It consists of millions of smaller domestic, academic, business, and government networks, which together carry various information and services, such as electronic mail, online chat, file transfer, and the interlinked web pages and other resources of the World Wide Web (www).

This network is spread all over the world. It is collection of interconnected networks. Hence it is called inter network" or in short "Internet". One can communicate with any other internet subscribers throughout the world through an Internet connection. Many companies including VSNL, Essar, Bharti Telecom, and MTNL provide Internet service in India. They are also known as Internet Service Providers (ISP).

The internet system is based on programs called "clients" and "servers". A program that provides a service is called a SERVER. A program that requests a service is called a CLIENT. The whole network is one large client/server system.

Any individual or organization can open an account with any Internet Service Provider (ISP) who will give an Account Number for monthly or yearly fees. Then the user may have access to the Internet and the e-mail through it. The user needs a computer, "modem" and a telephone line to access the Internet. In the user computer client programs are required. Those client programs carry out the commands by passing data back and forth to server programs. A server program might be in the next room, across the country or on the other side of the world. He can send and receive mails, surf to the World-Wide Web (www) or access his bank account through internet.

The Web is not internet. At times, people confuse the two terms that are related but not identical in meaning. The Internet is a collection of interconnected computer networks, whereas the Web is a collection of interconnected documents and other resources, linked by hyperlinks and URLs.

The World Wide Web is one of the services accessible via the Internet, along with various others including e-mail, online chatting etc. When you are on the Web, you are on the Internet but not the other way round. For example, those sending e-mail are not on the Web, unless they are sending e-mail via a Web browser.

*ISP:* Internet Service Provider. This is any company which provides Internet facility, either through dial-up or through leased lines. In India, some leading ISPs are Satyam, BSNL, VSNL, Airtel, etc.

*TCP/IP:* Transmission Control Protocol/Internet Protocol. Protocol refers to the rules of communication on a network. TCP/IP is the default protocol for communicating on the Internet. Any computer on the Internet needs to follow the TCP/IP protocol for communicating with any other computer on the Internet. TCP/IP is, in fact, a combination of two protocols:

- TCP - for transporting the data.
- IP - for locating a particular Computer.

IP is the name given to a 4-byte address, i.e., it is based purely on numbers. This address is unique for each computer on the Internet. Thus, using TCP/IP, we can specifically address any website on the Internet and send/receive data to/from it. Eg: the I.P. address for Raj Bhavan Bhopal is 70.87.15.202.

*URL:* Uniform (or Universal) Resource Locator: Though TCP/IP serves its role very well, it won't be easy remembering the TCP/IP address of every website. To simplify this, we have an English

equivalent of each web site's address. This English address is also unique and yet, it is easy to remember. This is known as the URL. It includes the protocol, the web server's address and the domain name, such as - http://www.rajbhavanmp.ind.in

*DNS:* Domain Name Service. When we type a URL into our browser and submit it to the ISP, the server at the ISP converts it into the corresponding IP address. After all, computers work best with numbers. Once the IP address has been obtained, the ISP's server will search for the IP worldwide. Once the desired server has been located, we can link to it through the IP. This mapping and conversion of the URL to the corresponding IP is provided by DNS.

*HTTP:* Hyper Text Transfer Protocol. This is the default way of sending data on the Internet. Through this protocol, hypertext can be transferred from one place to another. Though TCP/IP also helps in sending data, it operates at the network layer (of OSI). HTTP, in contrast, operates at the application layer (of OSI).

*WWW:* World Wide Web. The name given to the most popular service on the Internet. Through this, we can transfer any type of data from one place to another-text, graphics, audio, etc. This flexibility makes it so popular. In fact, its popularity has made it synonymous with the term Internet.

*FTP:* File Transfer Protocol. This Internet service is concerned with transferring files. Most of the file downloads we come across on the net – sample code, anti-virus updates, technical articles, media players, etc. – are based on the FTP protocol. In fact, most of the big Internet sites have dedicated FTP servers that deal with the downloading of files.

*Mail:* This service helps in sending e-mail on the Internet. E-Mail was probably the earliest use of the Internet. It continues to be one of the most commonly used services on the Internet even today. It utilizes the SMTP (Simple Mail Transfer Protocol).

## 11.3 HTML INTRODUCTION

HTML, or Hyper Text Markup Language, is a language for creating web pages. Most of the web sites we see on the Internet, whether big or small, have HTML at the core. HTML offers ease of use with a syntax that is too easy. Besides, there is no data-typing whatsoever in HTML. The concept of variables, so important in other programming languages, simply does not exist in HTML. It is a language with a highly simplistic structure. It is also highly forgiving- whenever we code in HTML, it won't give any error messages whatsoever, even if we violate the rules of HTML coding.

The HTML language - or technology - is based on the concept of tags. Tags are the keywords in HTML. In other languages we have statements such as if..then, do..while, etc. In HTML, all of our tasks are done through tags, which can be considered as the commands of HTML.

For coding in HTML, any text editor can be used, such as Notepad. Even MS-Word can be used to code in HTML. Specialised HTML editors are also available, such as MS-FrontPage, Netscape Composer, etc. Some free HTML editors are also available, such as Coffee Cup HTML (http://www.coffeecup.com), EasyHTML, etc. However, to really learn HTML in-depth, a simple text editor is the best option.

An HTML file is just a simple text file, saved with the extension '.htm' or '.html'. Once we navigate to a web page, this file is downloaded to our P.C. It is then executed by the browser on our PC - Internet Explorer, Netscape Navigator, Firefox, etc.

Here are some common tags in HTML and their meaning:

| Tag | Meaning |
|---|---|
| <HTML> | **Starts the HTML page** |
| <HEAD> | **Information for search engines and non-visual content** |
| <TITLE> | Title bar and search engine content. |
| <BODY> | Starts the visual portion of the web page |
| <B> | Makes the text bold |
| <I> | Makes the text italicized |
| <U> | Makes the text underlined |
| <S> | Gives the strike-through effect on text (~~strike-through~~) |
| <SUB> | Makes the text as a subscript ($H_2SO_4$) |
| <SUP> | Makes the text as a superscript ($4^2 = 16$) |
| <HR> | Draws a horizontal line |
| <BR> | **Inserts a line break** |
| <CENTER> | Centers the content horizontally |
| <H1>-<H6> | Header tags, rendering text in different sizes. |
| <A> | Anchor tag, for giving Hyperlinks, besides other things |
| <IMG> | Inserting an image |
| <OL> | Gives an ordered (numbered) list |
| <UL> | Gives an unordered (bulleted) list |
| <DL> | Gives a definition list, of keywords and their descriptions |
| <TABLE> | Shows content in a tabular format |
| <FONT> | Changes Font settings - type, size, colour |
| <FRAMESET> | Setting frames on a page |
| <FRAME> | Specifies content for a Frame |
| <SCRIPT> | Contains the scripting code |
| <STYLE> | Contains code for defining style sheets |
| <DIV> | Division tag, used as container to apply styles to code segments |
| <FORM> | Specifies an HTML form for accepting user data |
| <META> | Useful info for browsers and search engines, hidden from the user |

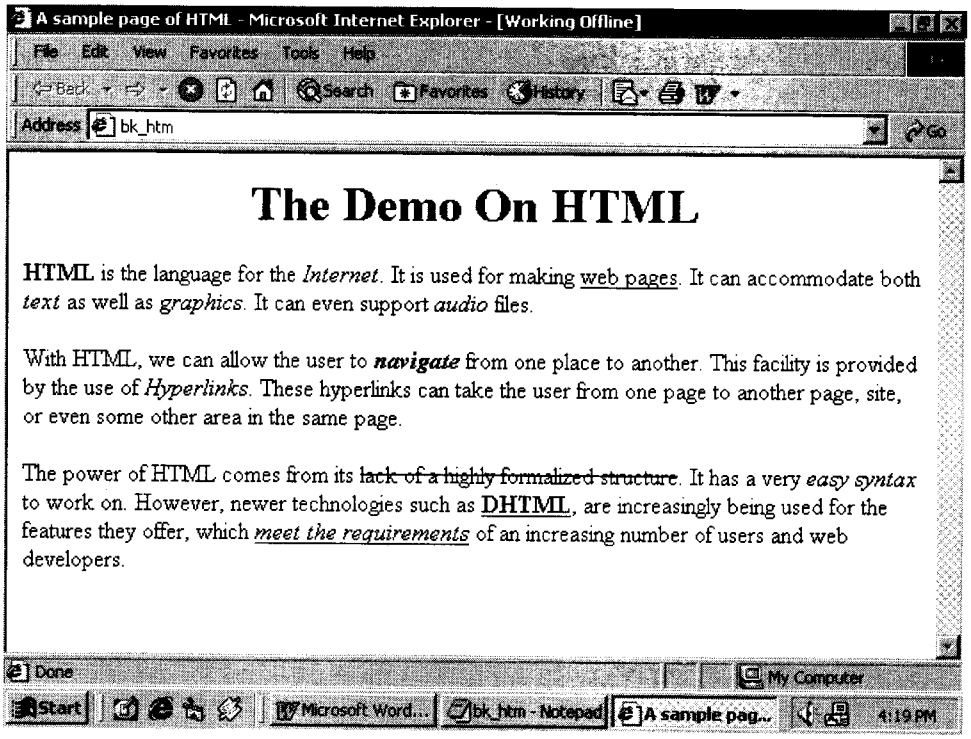
As an example, here is a small code of HTML.

Code Listing htm.1

```
<HTML>
<HEAD>
<TITLE>A sample page of HTML
```

```
</TITLE>
<BODY>
<H1 align="center"> The Demo On HTML</H1>
<B>HTML</B> is the language for the <I>Internet</I>. It is used for making
<U>web pages</U>. It can accommodate both <I>text</I> as well as
<I>graphics</I>. It can even support <I>audio</I> files.<BR><BR>
With HTML, we can allow the user to <B><I>navigate</B></I> from one place to
another. This facility is provided by the use of <I>Hyperlinks</I>. These
hyperlinks can take the user from one page to another page, site, or even
some other area in the same page.<BR><BR>
The power of HTML comes from its <S>lack of a highly formalized
structure</S>. It has a very <I>easy syntax</I> to work on. However, newer
technologies such as <B><U> DHTML</U></B>, are increasingly being used for
the features they offer, which <U><I>meet the requirements</U></I> of an
increasing number of users and web developers.
</BODY>
</HTML>
```

Here is a picture representing the output produced by this file. Observe the text formatting effects-italics, bold, underline, etc. produced by the code:



### 11.3.1 DHTML vs. HTML

HTML offered a lot of features for the creation of web pages and served our needs quite well. Originally intended as a 'mark-up language', its primary aim was to help in marking-up the contents. However, the needs of the users are changing and increasing day-by-day. They need special formatting and additional visual effects, which were not really the strength of HTML. For this reason, HTML has been customized to suit the changing needs of today. Formatting tags and some other elements have been incorporated in HTML.

Now, with too much of emphasis on presentation, HTML becomes somewhat bulky with all the content and the formatting information contained in the HTML page. The code becomes cluttered and hotch-potch. Moreover, inspite of all the efforts, HTML has not been able to keep pace with the expectations and requirements of today's designers. Also, it does not respond to the way the user is acting or behaving on the page.

The arrival of DHTML has changed all of that. DHTML makes web pages more lively, more colourful, and more responsive to the user. We might have seen the following scenarios:

- A web page which keeps changing in colour or content as we move the mouse around the page.
- An image which suddenly pops up out of nowhere on a web page when the mouse reaches at certain spot.
- A message which gets displayed only when we take the mouse to a particular place.
- Some buttons on a web page, which keep changing in colour as the mouse passes over each one of them.

If we have seen any of these, we have already seen DHTML at work. DHTML is the technology which makes the web pages come alive, pack more punch, and add an element of interactivity or responsiveness to the end-user. We can even use it to connect to a database and increase the power and utility of our web pages.

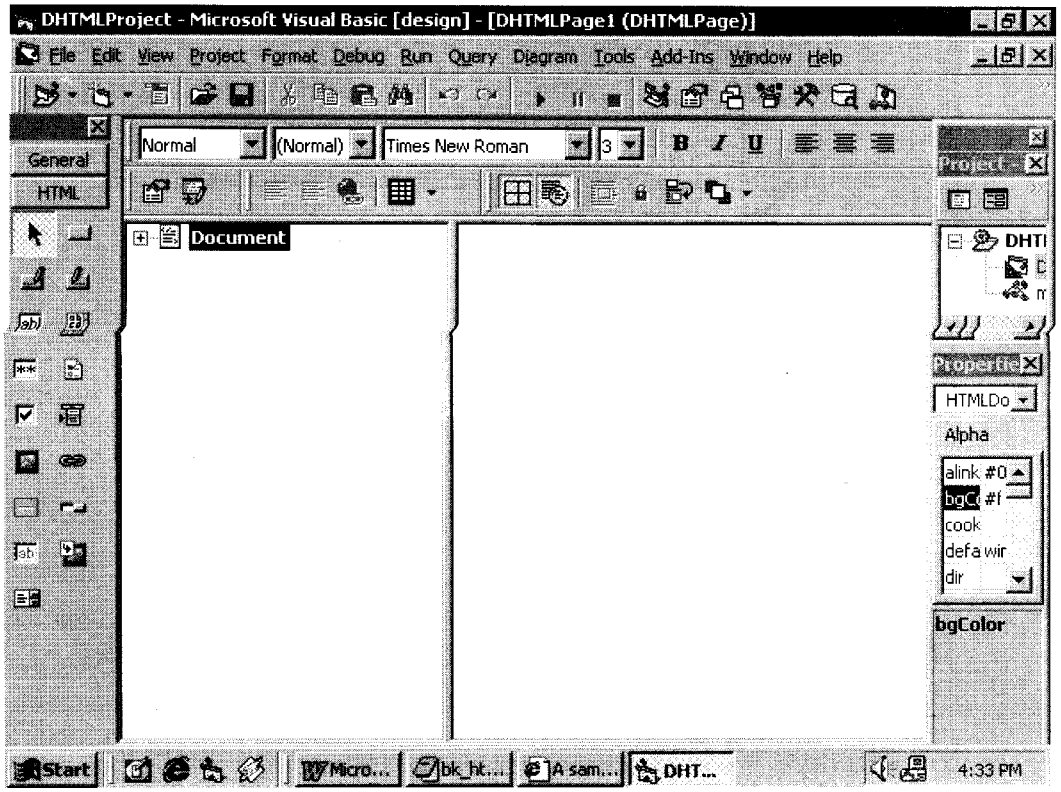
The power of DHTML comes from two interesting features - Style Sheets and Scripting. Let us see what they are.

## 11.4 INTERNET APPLICATIONS USING VISUAL BASICS

*Starting the DHTML Project*

To start with DHTML in VB,

1. Start VB as usual.
2. Select 'DHTML Project' from the VB project options start-up box. In the opening screen, we see icons for a Code Module and a DHTML Page Designer in the Project Explorer.
3. Double-click on the Designer in the Project Explorer. The following screen opens up:



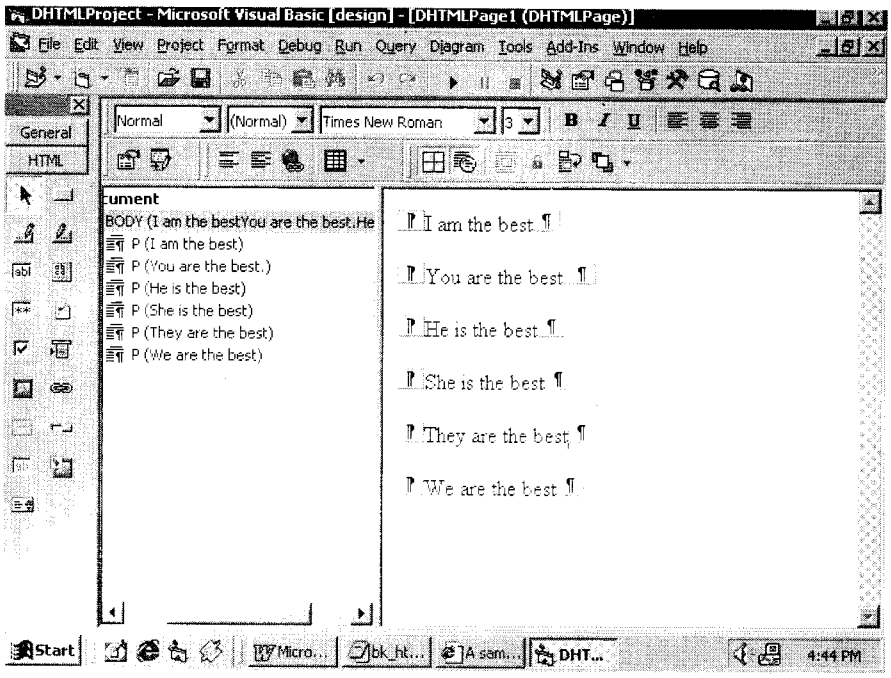

This designer will help develop the front-end for our DHTML application. Some elements of the ToolBar are very easily recognizable, such as the text-alignment, bold, italic, font-size, font-type options, etc. The properties window and the toolbox too are easily recognizable. The toolbox, though, offers a different set of controls to work with the controls for working on HTML pages. The center of the screen offers us two frames. The left frame is the treeview, where we can see a hierarchical view of our web page. On the right is the design frame, where we

- type in our page's content, and/or
- add controls for interacting with the end-user.

This way, the right frame is like the Form of VB we know. We will be working with the right frame for the most part, when working in DHTML.

*Using the DHTML Page Designer*

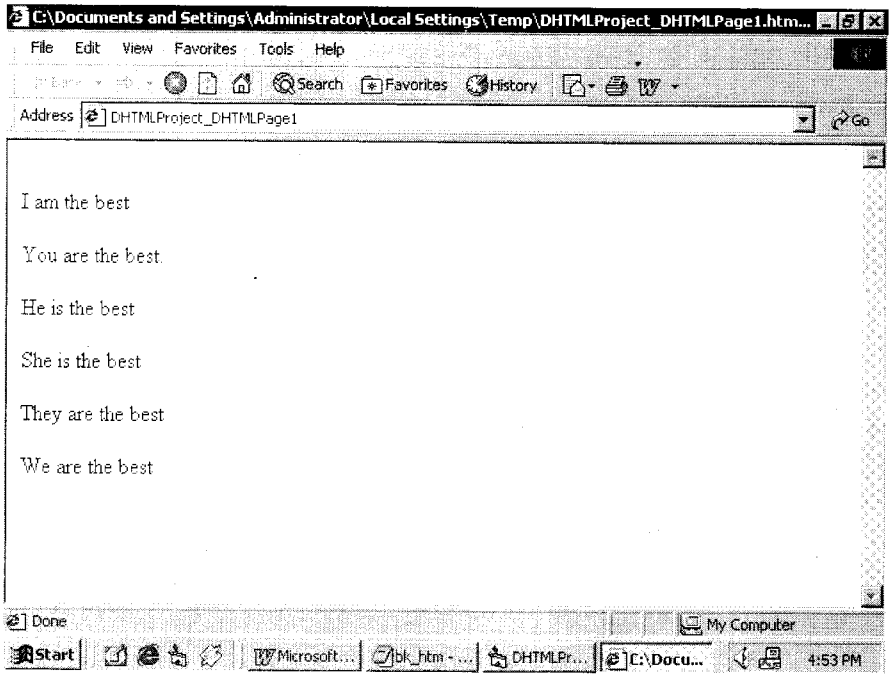
To get our feet wet in the pond of DHTML, first we will design a simple page with no DHTML. It will just take on some plain text and we will be seeing the results of plain and simple HTML. Here is how we go ahead: In the right frame, click at the top-left and start typing the content you want to be included in the web page. Once we have typed a complete paragraph, press the 'enter' key. The [ ] delimiters will appear around our paragraph, marking the start and end of the paragraph. This is similar to what we see in MS-Word. Now enter some more text for more paragraphs. Let us say our DHTML designer screen looks as follows:




Observe that the left side of the designer - the treeview - is displaying our web page hierarchically.

### *Running the DHTML Project*

Now we run our DHTML project, in the usual VB manner. The debugging tab appears, seeking information on how to start the project. Leave the screen at the default values. Anyway, these are the same entries we make when we run our custom Active-X control for the first time. Leave the values at the default and click on 'OK'. We see the page opening in our browser, as shown here:



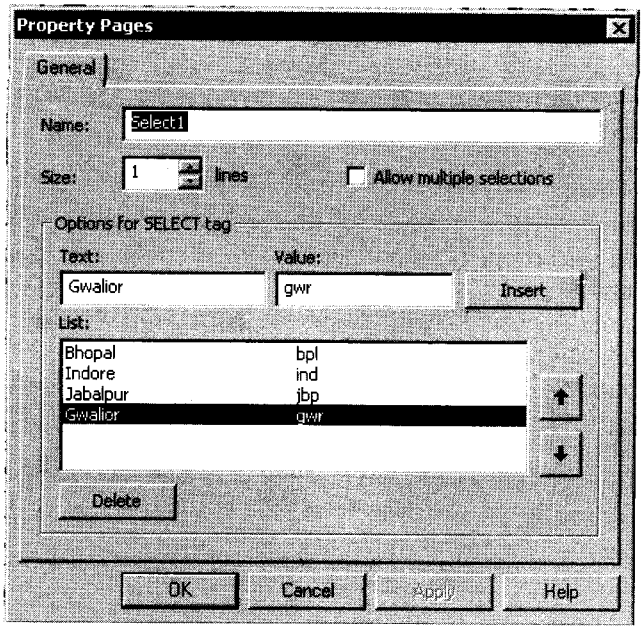
### *User Interface Development*

Our basic HTML page is now ready and we have seen it running in Internet Explorer, too. If required, we can use the toolbar provided by the DHTML environment to add text formatting features such as bold, italics, font size, font-type, etc, as we commonly do in MS-Word.

### *Using HTML Form Controls*

The toolbox on the left shows the controls that can be used on an HTML Form. One common attribute of all the elements is that they are identified by the Id attribute. The Name property is not applicable in DHTML. Also, the Caption property is replaced by Value.

- We see four buttons in the ToolBox:
  - The 1st button is the simple commandbutton we know.
  - The 2nd button is the 'Submit' button, used in association with an HTML Form. Once we fill our data in the Form and click on the 'Submit' button, the data is sent to the server for further processing.
  - The 3rd button is the 'Reset' button. Pressing on this button clears the contents of our Form- it does not send the data to the server.
  - The 'FileUpload' button is for 'uploading' a file to the server. For example, the attachments we send alongwith our E-Mail.
- The 'TextField' is the simple TextBox we know.
- The 'TextArea' is a multi-line box, commonly used to type in/display long text.
- The 'Select' option is the drop-down combo of the HTML world. It can also serve as a standard listbox, if we set the 'Size' property to a value greater than 1. To enter the list elements of 'Select', click on 'Custom' in the properties window. The following screen pops up, helping us enter text of list elements and their corresponding values:



In the dialog box enter the Size, i.e., the number of elements that should be visible in the 'Select' control at any time. Now set the Text and the Value by clicking on the 'Insert' button.

A size of more than 1 will make our 'select' control change from a drop-down list to a simple list.

Text is shown to the user, Value is sent to the server once user clicks on 'Submit'. Use only few characters for the value, to minimize data transfer between the client and server.

### 11.4.1 Event Handling

Any element we create on the page can raise an event, provided it has an Id attribute. Thus, to trap a paragraph's onclick event, we assign it an Id and code as required. The DHTMLpage and also the BaseWindow (representing the browser) too raise their own events. The lostfocus event of traditional VB is replaced by onblur, the click event is replaced by onclick, the load event by onload, and so on. Of course, some events are not available in VB, such as onmouseout (the mouse goes out of a named area), and onmouseover (the mouse enters a named area).

*Changing the Content Dynamically*

Continuing with DHTML, we come to another vital aspect - run-time text display. We should be able to generate some text on an as-is-required basis, when the user passes the Mouse over a particular area. This is another power of DHTML, with which we can generate web pages on-the-fly, as and when required.

To generate dynamic content, we create a 'blank paragraph' - a paragraph with no text. To create such a paragraph, come to the first para and press the Enter key twice. Create this paragraph just under the first paragraph 'para1' and give it the name 'para_blank'. Now switch to the code window and write this code for the onmouseover event of 'para1'. (assuming 'para1' is the Id for our 1st paragraph).

Code Listing dht.1

```
PRIVATE SUB PARA1_ONMOUSEOVER( )
para_blank.INNERTEXT = "This is a demo of the innertext OF DHTML"
END SUB
```

Run the program and bring the mouse anywhere on 'para1'. We will see the text given above appear from nowhere, under para1. This is the power of DHTML.

In addition to simply inserting plain text at run-time, DHML also offers the facility of adding HTML-formatted text. This will further liven up the display. This feature is offered by the innerHtml property. To see this, add two more blank paragraphs to the web page just under 'para_blank', with the Id attributes set to 'para_blank1' and 'para_blank2'. Now code as follows:

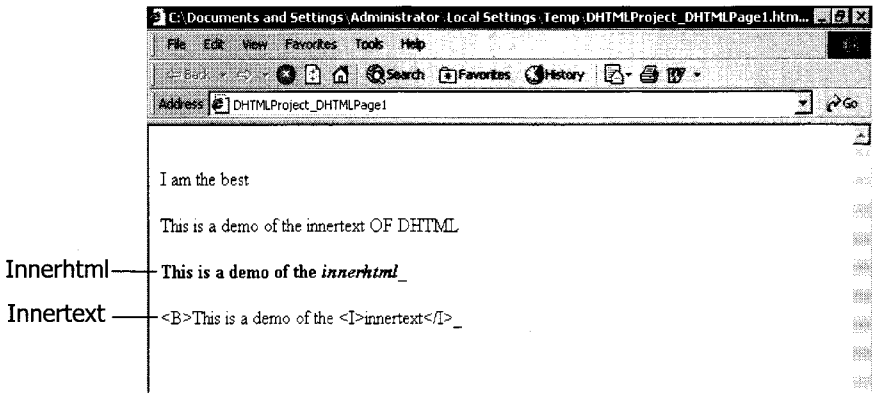
Code Listing dht.2

```
PRIVATE SUB para1_ONMOUSEOVER()
para_blank.INNERTEXT = "This is a demo of the innertext OF DHTML"
para_blank1.INNERHTML = "<B>This is a demo of the <I>innerhtml</
I>_ feature of DHTML</B>"
para_blank2.INNERTEXT = "<B>This is a demo of the <I>innertext</
```

```
I>_feature of DHTML</B>"
END SUB
```

Run the program to see the effect of innertext and contrast it with the result of innerhtml:

- With "innerhtml", text has been displayed with HTML formatting.
- With "innertext", text has been displayed as simple text, without applying any HTML formatting. The HTML tags have been displayed as plain text on the page.




Now we make a small program to add two numbers in textfields and display the result in another textfield. Start a new DHTML Project and open the DHTML Page Designer. Now add three textfields to the designer, alongwith a button. Give the names tf1, tf2, tf3 and butt1 to the controls. Now come to the code window and code for the 'onclick' of 'butt1' as shown here:

Code Listing dht.3

```
PRIVATE FUNCTION butt1_ONCLICK( ) AS BOOLEAN
tf3.VALUE = CINT(tf1.VALUE) + CINT(tf2.VALUE)
END FUNCTION
```

Run the program, entering a number in 'tf1' and 'tf2'. Then click on 'butt1'. The sum of the textfields 'tf1' and 'tf2' will be displayed in the textfield 'tf3'.

DHTML does not offer the "text" property. Instead, the DHTML controls have the value property with the same meaning.

The val function might not work with the code given above. It can raise an error, depending on the version of the Browser.

## 11.4.2 Applying Style Sheets

Now for the styling - adding special effects to the page. Let us say that when we bring the mouse over the 1st paragraph, we want the following effects:

- The background colour of the paragraph changing to black.
- The text colour changing to white.
- The font type changing to 'arial'.
- The font size changing to 20.

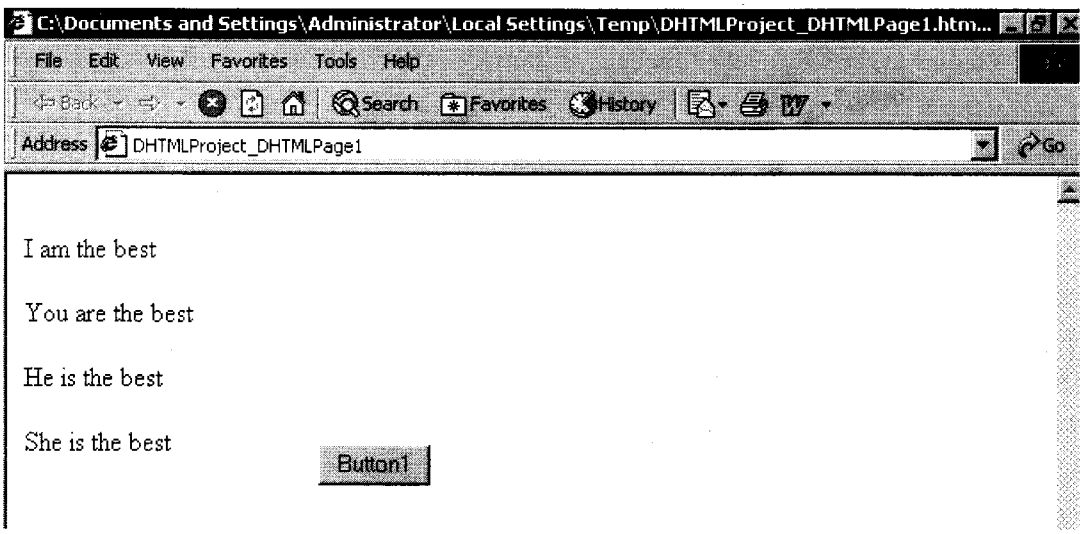
- The paragraph showing a border.
- The paragraph getting center-aligned.

To do so, select the first paragraph and give it the id of 'para1' in the properties window. Now come to the code window and code for the onmouseover event of para1:

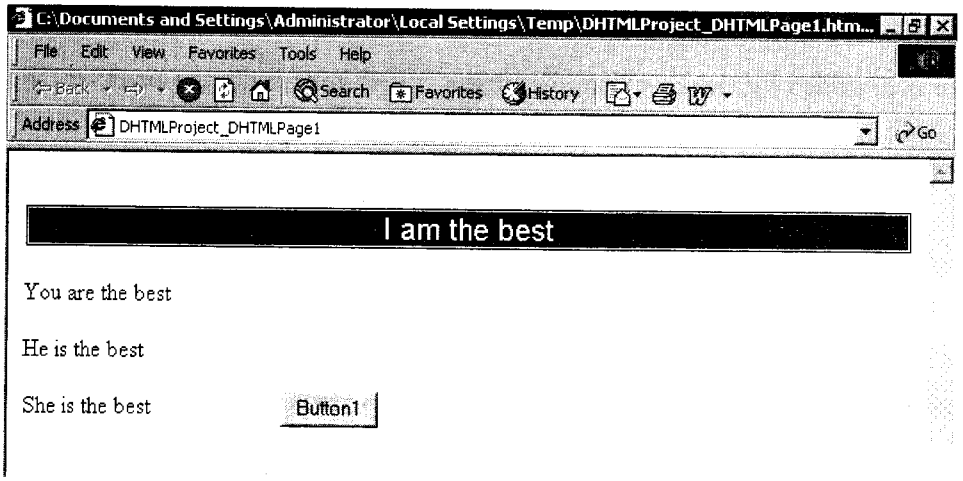
Code Listing dht.4

```
PRIVATE SUB para1_ONMOUSEOVER( )
para1.STYLE.BACKGROUNDCOLOR = "black"
'set the background color to black
para1.STYLE.COLOR = "white"
'set the foreground color to white
para1.STYLE.FONTFAMILY = "arial"
para1.STYLE.FONTSIZE = 20
para1.STYLE.BORDER = "double"
para1.ALIGN = "center"
END SUB
```

When we run the program, we get the typical output as seen here. Nothing special here.




Now bring the mouse over the para named 'para1' and see the change in the result, as per the following figure. Especially note the effect on the text 'I am the best' (this is the 'para1' of our DHTML page):

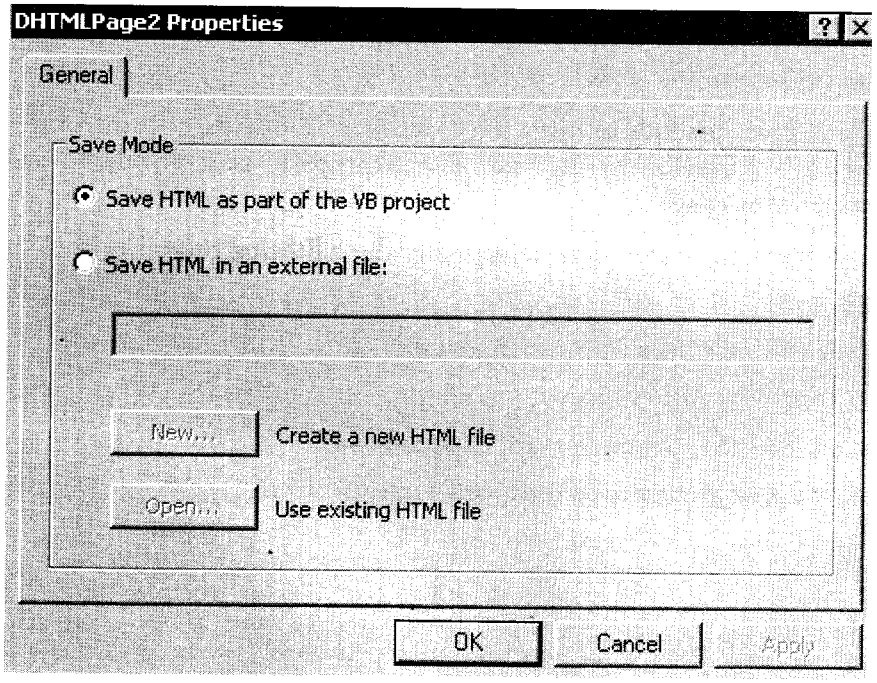



Style is both a property as well as an object in itself, since it is having its own properties such as colour, border, etc.

### 11.4.3 Adding a New DHTML Page

From the 'Project' menu, select 'add DHTML Page'. The Page Properties dialog box opens up.

- The option 'Save HTML as part of the VB project' will 'fully integrate' the new page with the current project. It can be opened from within the project.
- The option 'Save HTML in an external file' will depend on the two buttons given at the bottom:
  - ❖ Clicking on 'New' saves the new page as a simple HTML file. It can be opened even outside the project.
  - ❖ Clicking on 'Open' adds an existing HTML page to our project.



*Navigating from One Page to Another*

This feature is provided by the BaseWindow object, which represents the browser. We code for it as follows:

```
BASEWINDOW.NAVIGATE (url)
```

For example, to go to page number 2, we write the following code (if we saved our page with 'Save HTML as part of the VB project'):

```
BASEWINDOW.NAVIGATE ( "dhtmlproject_dhtmlpage2.HTML" )
```

*Maintaining State Information*

HTTP is a 'stateless protocol'. The server does not keep track of clients between calls. Every new request for a page is treated as a request from a new client, even if the client is coming with a request for the 6th time in just under 5 minutes. Thus, an entry made once by the user will be 'forgotten' by the server. To overcome this statelessness of HTTP, we make use of the putproperty and getproperty properties:

- The putproperty method helps write a value.
- The getproperty method helps read a value.

The syntax of the two methods is:

```
PUTPROPERTY (objdocument AS HTMLDOCUMENT, strName AS STRING, vntvalue)
```

where

- objdocument is the Document we are referring to.
- strName is the name of the property to be created.
- vntvalue is the value we want to give to our property.

variable_name = GETPROPERTY [objdocument AS HTMLDOCUMENT, strName AS STRING]

where the parameters mean the same as for putproperty.

Normally, we use these methods to carry over values from one page to the other. The putproperty stores the value to be taken in the browser. The getproperty gets the value stored in the browser, coming from the previous page.

In the page where we want to store a value, we enter the following code:

Code Listing dht.5

```
PRIVATE SUB butt1_ONCLICK ( ) AS BOOLEAN
PUTPROPERTY BASEWINDOW.DOCUMENT, "prop1", tf1.VALUE
'put the value of the property in the browser
BASEWINDOW.NAVIGATE "dhtmlproject_dhtmlpage2.html"
END SUB
```

In the page where we want to retrieve the value, we code as follows:

```
PRIVATE SUB butt1_ONCLICK ( ) AS BOOLEAN
tf8.VALUE = GETPROPERTY(BASEWINDOW.DOCUMENT, "prop1")
'get the value of the property from the browser
END SUB
```

If we get error no.13, 'type mismatch' when trying to maintain state, it is due to Internet Explorer version 5.0. To avoid it, change the code from basewindow.document to document.

### *Deploying a DHTML Application*

The DHTML project will be compiled as a DLL file. Once we create a distribution package for it, the package will comprise of:

- Main DLL file with compiled application code
- VB6 runtime files
- HTM files that are part of the application
- Additional files, such as data files, images, etc.

**Check Your Progress**

Fill in the blanks:

1. IP is the name given to a ..... address, i.e., it is based purely on numbers.
2. The HTML language is based on the concept of .....
3. The ..... tab appears, seeking information on how to start the project.
4. DHTML does not offer the ..... property.

## 11.5 LET US SUM UP

Internet Service Provider provides Internet facility, either through dial-up or through leased lines. In India, some leading ISPs are Satyam, BSNL, VSNL, Airtel, etc. HTML, or Hyper Text Markup Language, is a language for creating web pages. Most of the web sites we see on the Internet, whether big or small, have HTML at the core. HTML offered a lot of features for the creation of web pages and served our needs quite well. Originally intended as a 'mark-up language', its primary aim was to help in marking-up the contents. However, the needs of the users are changing and increasing day-by-day. To get our feet wet in the pond of DHTML, first we will design a simple page with no DHTML. It will just take on some plain text and we will be seeing the results of plain and simple HTML. HTTP is a 'stateless protocol'. The server does not keep track of clients between calls. Every new request for a page is treated as a request from a new client, even if the client is coming with a request for the 6th time in just under 5 minutes.

## 11.6 KEYWORDS

*ISP:* Internet Service Provider

*TCP/IP:* Transmission Control Protocol/Internet Protocol

*URL:* Uniform (or Universal) Resource Locator

*DNS:* Domain Name Service

*HTTP:* Hyper Text Transfer Protocol

*WWW:* World Wide Web

*FTP:* File Transfer Protocol

*Mail:* This service helps in sending e-mail on the Internet.

*SMTP:* Simple Mail Transfer Protocol

*HTML:* Hyper Text Markup Language

*DHTML:* Dynamic Hyper Text Markup Language

## 11.7 QUESTIONS FOR DISCUSSION

1. What is internet? What are major terms used in setting up a internet connection?
2. Differentiate between HTML And DHTML.
3. Discuss the steps involved in staring and running a DHTML project.
4. Explain the event handling in DHTML.

**Check Your Progress: Model Answers**

1. 4-byte
2. Tags
3. Debugging
4. text

## 11.8 SUGGESTED READINGS

Cornell, G., *Visual Basic 6 from the Ground Up*, Tata McGraw Hill.

Murray et.al, *Visual C++ Handbook*, 2nd Edition Osborne, McGraw Hill, 1996.

Evangelos Petroutsos, *Mastering Visual Basic 6*, BPB Publications.

LESSON

# 12

# ACTIVEX DOCUMENTS

CONTENTS



## 12.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Create an ActiveX control
- Add properties to an ActiveX control
- Add events to an ActiveX control
- Add methods to an ActiveX control
- Create EXE and DLL forms of an ActiveX control

# 12.1 INTRODUCTION

ActiveX is a set of reusable Components that can be created and utilized by several applications. ActiveX uses the Internet technology to assist in creating compact and reusable applications that can be deployed via the Internet or a corporate Intranet.

ActiveX provides a familiar client-server infrastructure to run your application. This lesson introduces you with few important ActiveX controls that are provided by Microsoft. Controls like Coolbar, Flat Scrollbar, ImageCombo, DateTimePicker control have been released with Visual Basic's sixth version.

# 12.2 ACTIVEX CONTROLS

ActiveX controls are the custom controls that can be added to the ToolBox and used in several VB applications. ActiveX controls can also be used in other ActiveX compliant programs. These controls can also be embedded and distributed through HTML web pages.

## 12.2.1 Image List Control

While working with Toolbar or Coolbar controls you need to assign an image to each button. Instead of loading each image into the memory before using, it is easier to store the reference of all images in a single place.

The Image list control works behind the curtain i.e., it is not visible at run time and gives another way to store a group of images in a single place. The key to work with an Image list control is ListImage object and the ListImages collection. The ListImages collection specifies the images stored. The main properties for an ImageList control are its custom properties, which are shown by a dialog box. You can set pictures in the ImageList control at design time and use their index or key to refer. The image tab on this dialog box gives you a convenient way to add images at design time.

You can refer to the item stored in the ListImages collection by either an index number or a key. For example: To use Image from an ImageList control

```
Set Picture1.Picture=ImageList1.ListImage(1).Picture
```

This example fills the Picturebox with the first image in the Image list Control. We shall use this control in the coming example for other controls.

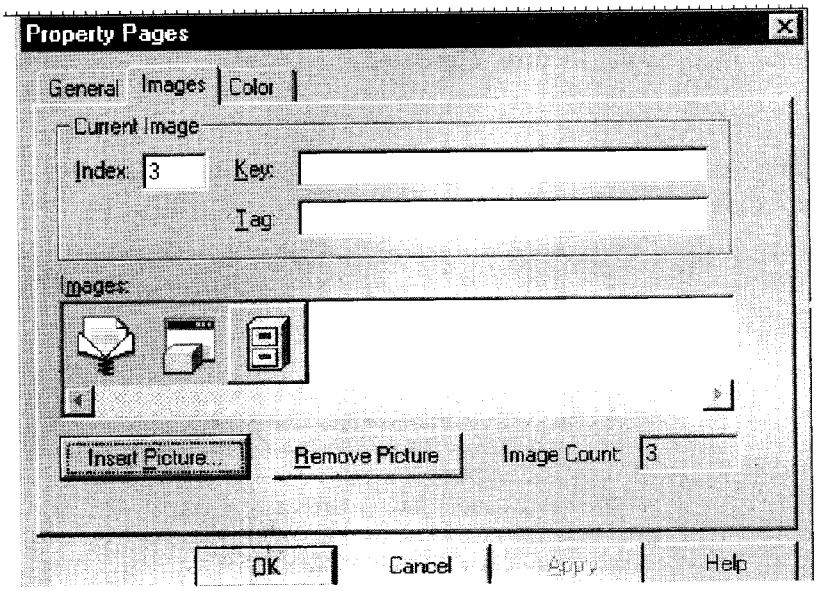



Figure 12.1

## 12.2.2 Toolbar Control

Toolbar has become one of the most important tools for providing an easy interface to the users. The Toolbar control provides easy access to options available in your applications. To add design time properties to the controls select its custom properties from the property box.

A toolbar control contains a collection of Buttons objects used to create a toolbar. A toolbar contains buttons that correspond to items in an application's menu, providing a graphical interface for the user to access an application's most frequently used functions and operations. You can even add an image in a button. You need to add an ImageList control on the same form, to add the images in the toolbar at design time.

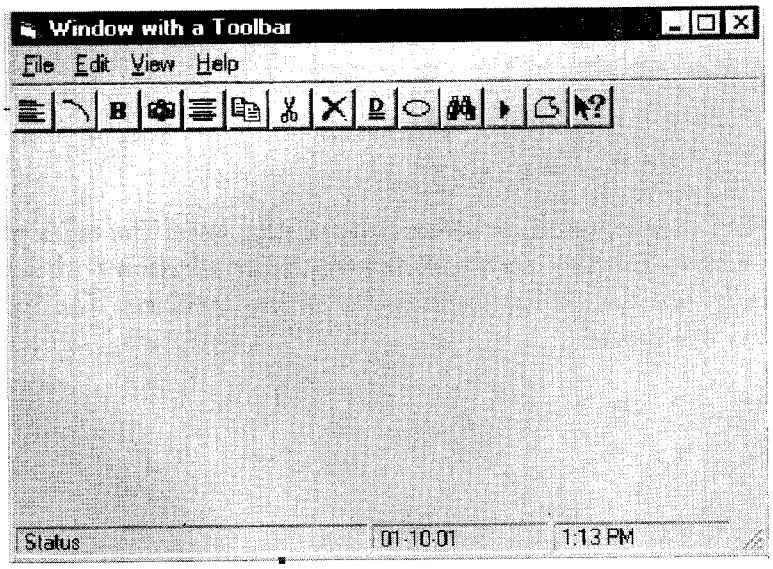



Figure 12.2

You can add buttons at design time in the toolbar using its property page. The custom property page has three tabs, General, Buttons and Picture. The description of these tabs is given below:

*General Tab*

MousePointer property sets the kind of Mouse pointer to be used. ImageList property sets the name of the ImageList control to be used for obtaining the Pictures.

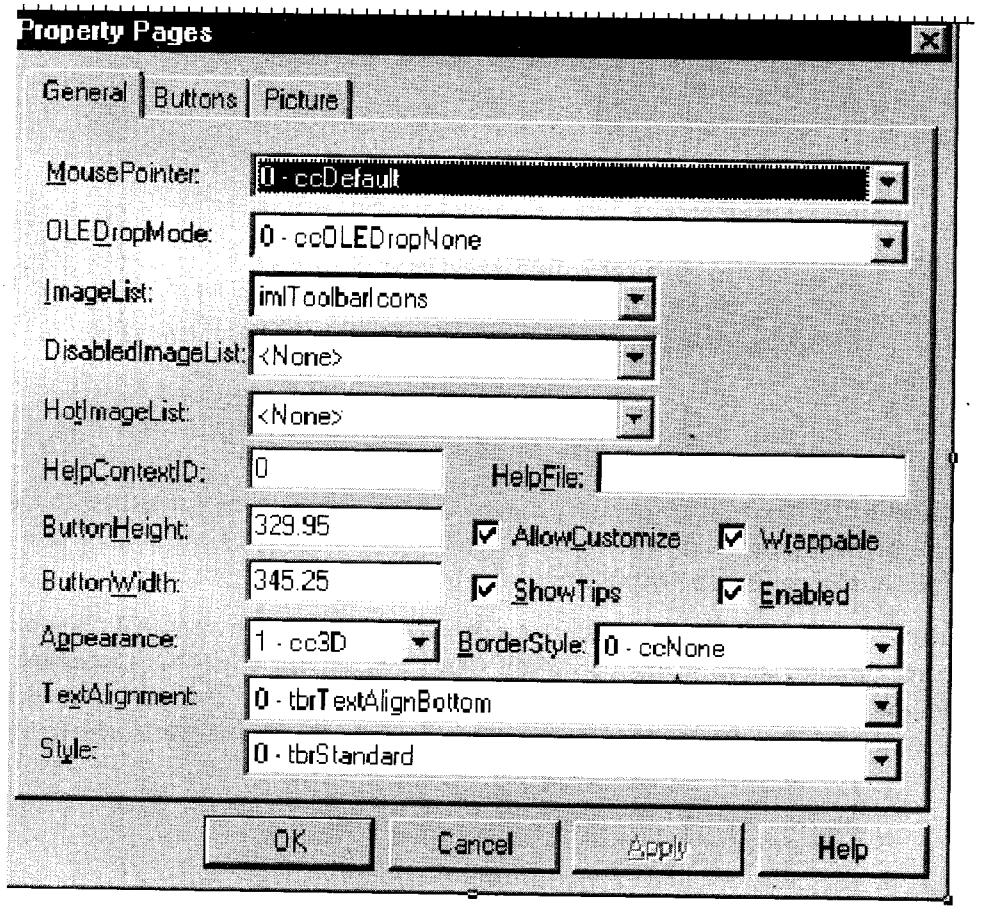



Figure 12.3

*Buttons Tab*

This tab allows you to add new buttons or remove the existing ones. The index parameter returns the index of the button added. The Key parameter lets you refer a button with a different name. The Caption parameter is the text that will appear beneath the button object. The Style property lets you specify how the button will appear.

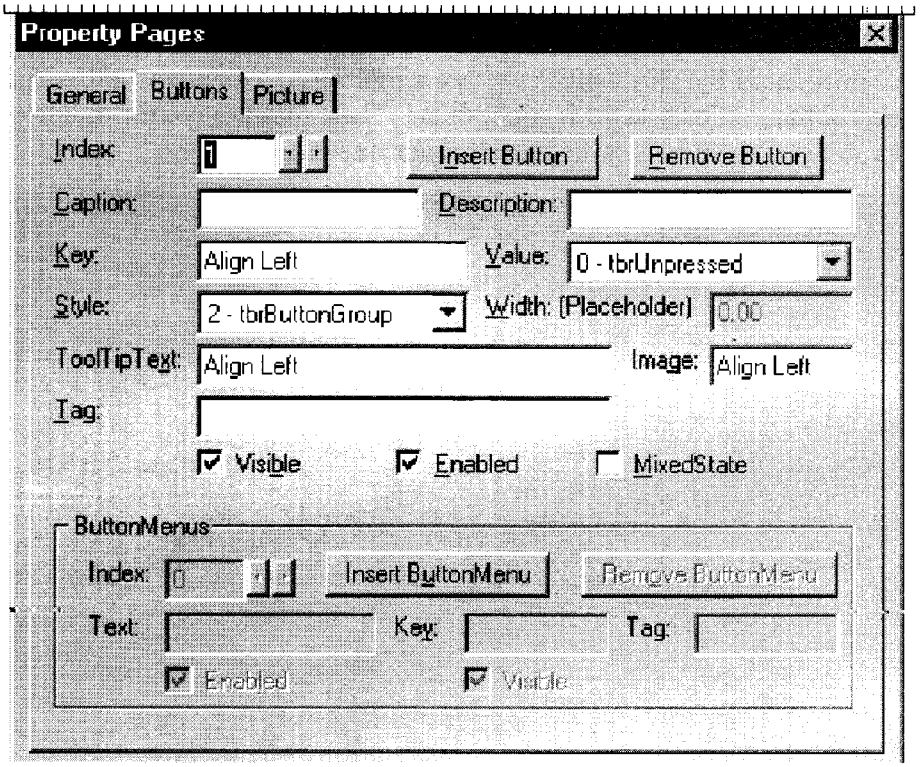



Figure 12.4

Settings for the Style property are:

| | |
|---|---|
| tbrDefault | The button is a regular Push button. |
| tbrCheck | The button is a check button. |
| tbrButtongroup | The button is a part of a button group. As Alignment buttons in MS-Word's standard toolbar. |
| tbrSeparator | The button is simply a Seperator. |
| tbrPlaceHolder | The button is a variable width Seperator. |

Image parameter is the index or key that describes which images you want from the previously associated ImageList control. If AllowCustomize property is set to TRUE the toolbar at run time, on double clicking opens a dialog box to customize the toolbar. You can remove or change the position of the buttons at run time using this dialog box. Use Add method to add the buttons at run time.

Syntax:

SetButton object = Name of Toolbar.Add([Index], [Key], [Caption], [Style], [Image])

You can also first add the button and then describe it's properties.

```
e.g., Toolbar1.Add
    Toolbar1.Buttons(2).Caption="Second Button"
    Toolbar1.Buttons.Add
    Toolbar1.Buttons(3).Caption="Third Button"
```

### 12.2.3 Coolbar Control

The new control introduced by Microsoft to replace a toolbar control is a Coolbar control. Coolbar enables you to add sliding toolbars to your applications. These toolbars consist of one or more bands, with the bands existing on the same row or starting from a new row.

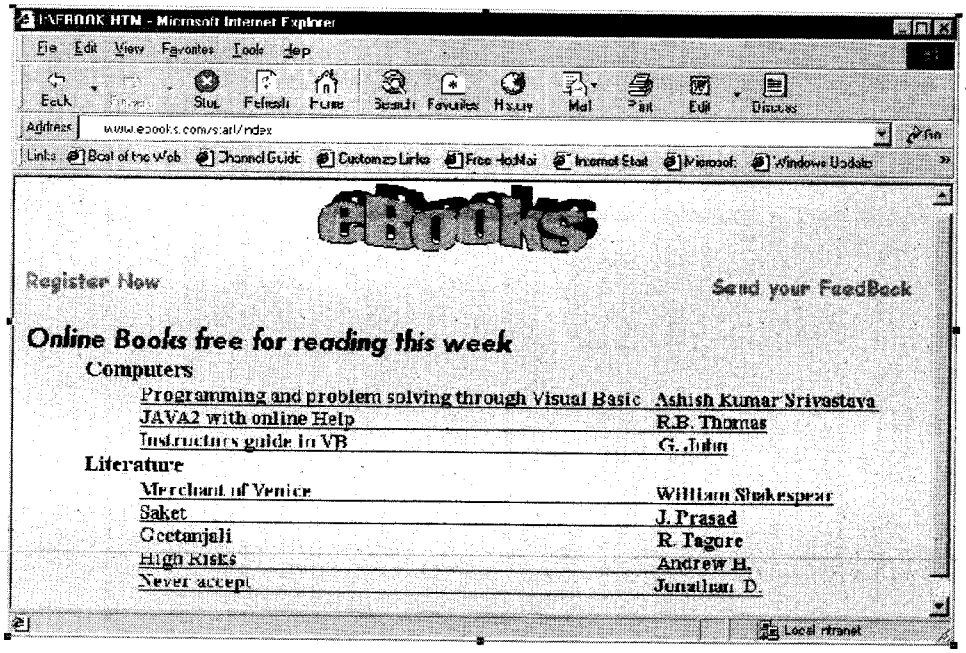



**Figure 12.5**

An example of Coolbar control can be seen in Internet Explorer's toolbar. You can add menu items, shortcut Buttons, List Box, Combo Box etc. to it. Each band has a move handle which is a vertical line that an be found on the left most side. Coolbar, basically is used as a container and can include just any kind of control. For example you can add a Toolbar, List control, a Text box or even a Combo box.

Some of its important properties are listed below:

*Coolbar Control's Properties*

- *Orientation:* This property specifies the orientation of the CoolBar. This property directly affects the control's bands. Various settings for orientation property are:

| Value | Constant | Description |
|---|---|---|
| 0 | CC3OrientationHorizontal | Horizontal Orientation |
| 1 | CC3OrientationVertical | Vertical Orientation |

- *Picture:* The Picture property is used to specify the picture file name for overall background image for the overall CoolBar control. Same picture can be used by different bands in the CoolBar control by setting their Use CoolBarPicture property to True.
- *BandBorders:* The BandBorder property is a boolean property that specifies whether or not the borders between the bands should be displayed.

- ***VariantHeight:*** Size of the bands can be made of equal height by setting the Variant Height property to False. When VariantHeight property is set to False, the height of the Bands is based on the maximum MinHeight property of all the bands. Default value for VariantHeight property is True.
- ***Align:*** Align property specifies the alignment of CoolBar control on its container object. Different settings for Align property are discussed below. When values for this property, are used the CoolBar is not movable. To make it movable set this property to 0 (vbAlignNone).

| Value | Constant | Description |
|---|---|---|
| 0 | VbAlignNone | No Alignment specified |
| 1 | VbAlignTop | Control is set at the Top Border of the container object |
| 2 | VbAlignBottom | Control is set at the Bottom Border of the containerobject. |
| 3 | VbAlignLeft | Control is set at the Left Border of the container object. |
| 4 | VbAlignRight | Control is set at the Right Border of the container object. |

- ***ImageList:*** ImageList property is used to specify the name of the control containing images, which can be displayed next to the band's move handles.

The CoolBar control has one unique event **HeightChanged()** beside the standard Click(), MouseMove() or DragDrop() events. The HeightChanged() event triggers when there is a change in the height by the user or bands are rearranged and can be used to manipulate object in the CoolBar control. After learning about the properties and events of the CoolBar control, let's move to the Band and Bands collection. As we already know a coolbar control consists of one or more different sections, known as Bands. The Band objects together form a Bands Collection. Methods that apply to the Bands collection are Add, Clear, Item similar to rest of the collection. The Band has no events or methods, but has only unique set of properties. Most of the Band object properties are used to specify the alignment and appearance of the Band. The different properties of the Band object are:

| Property | Description |
|---|---|
| AllowVertical | Specifies the display of control when the Coolbar's orientation is vertical. |
| Caption | Caption of the Band. For vertical CoolBars, the Caption appears below the band's move handle. |
| Child | Specifies the name of the control that will be contained by the band. |
| Key | Used to assign a unique name to the band for later use in coding |
| MinHeight | Minimum height of the Band. |
| MinWidth | Minimum width of the Band. |
| NewRow | Specifies whether the band will appear in the new row in the CoolBar. |
| Position | Returns the numerical position of a band object. Bands are numbered from Left to Right or Top to Bottom. |
| Style | Specifies whether a band can be resized or not. |

## 12.2.4 ImageCombo Control

ImageCombo is just another Combo Box with an additional facility of adding images along with the text.

Each item can have two images, one when the item is selected and the other when it is not selected. The ImageCombo control also enables you to indent items in the list. This facilitates the arrangement of items in a hierarchical manner. Each item in an ImageCombo control is called **ComboItem** object. These ComboItem objects together form a collection known as **ComboItems.**

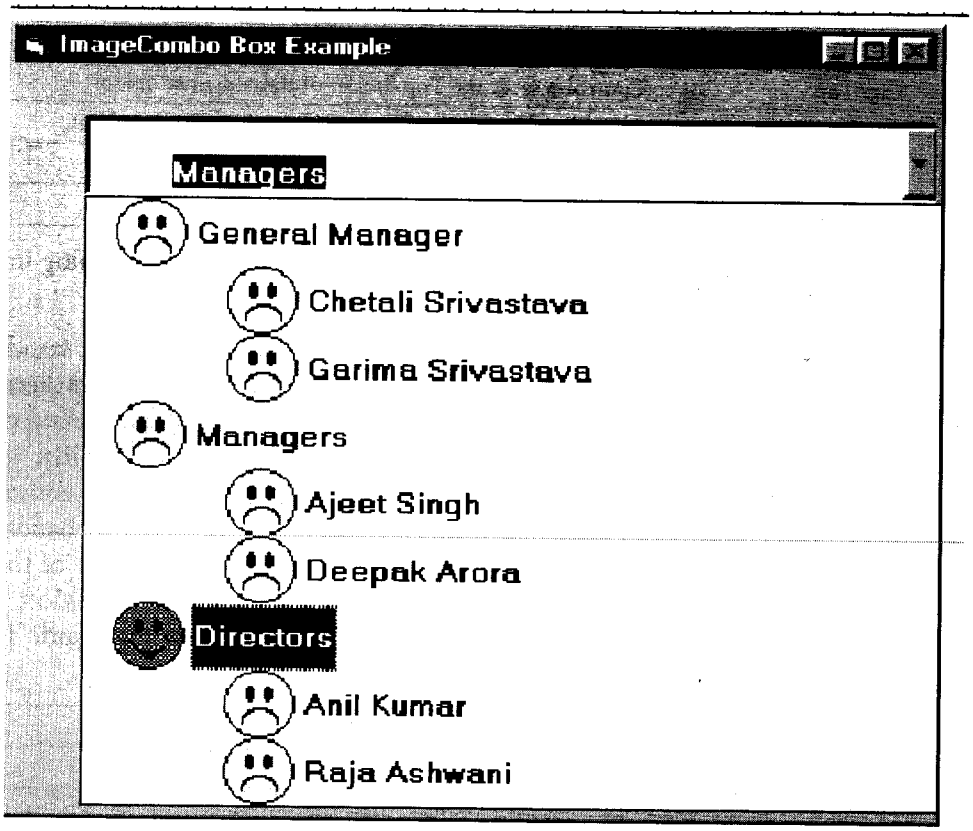



Figure 12.6

- The Additem and RemoveItem methods do not work with ImageCombo unlike the ComboBox.
- ImageCombo uses Add and Remove methods to add or remove a ComboItem object to or from the ComboItems collection.

*ImageCombo Properties, Methods and Events*

Since ImageCombo control is provided with ComboItem object, therefore most of the handling is done through these objects. However, there are few properties and methods associated with the ImageCombo. An awareness with the ComboBox control will let you work with ImageCombo box easily. Now lets look at the following properties of the ImageCombo:

### *ImageList*

ImageList property keeps a reference to the ImageList control. ImageList control is used to store the references to number of picture or icon files for later use. These files are then referenced with their index number or key name. ImageList property automatically shows the names of the ImageList controls available on the form.

### *Indentation*

The Indentation property can be used to specify a default indentation for any ComboItems object, belonging to ComboItem's collection. This property can be set while adding a new ComboItem object to the ComboItems collection.

- Example :

```
ImageCombo1.ComboItems.Add(1, "C1", "Fruits",5,6,1)
ImageCombo1.ComboItems.Add(2, "F1", "Apple",1,2,3)
ImageCombo1.ComboItems.Add(3,"F2", "Mango",3,4,3)
```

The above example adds a Headng "Fruits" with key name C1 with indentation factor 1 and then add Apple and Mango as sub contents with indentation factor 3 to each.

### *SelLength, SelStart and SelText*

**SelLength**, **SelStart** and **SelText** properties are the three properties used with Rich Text Box control and with the ComboBox control. **SelLength** property returns the number of characters selected i.e., the length of the text of ComboItem object selected. **SelStart** property returns or sets cursor position. **SelText** property returns or sets the text that has been selected.

### *GetFirstVisible, SetFirstVisible Methods*

Applies to the ComboItem object and specifies whether a particular list item is selected or not. Returns boolean value. These methods help you to get or set the first ComboItem object visible in the ImageCombo.

### *Important Events and Methods of the ImageCombo Control*

Click(), Change() and Validate() are the important events used to work with the ImageCombo. These events trigger as they trigger in standard ComboBox. In order to add a new item to the ImageCombo use Add method of the ComboItems collection. The Add method has the following syntax.

Add ([Index], [Key], [Text], [Image], [SelImage], [Indentation])

Each argument in the Add method is optional. These arguments correspond to the ImageCombo item's properties and thus, new properties can be set while adding a new member to the collection at design time. Similarly, you can remove an individual item using the Remove method.

Remove method removes only one item at a time. The argument of the Remove method is only index number of the item to be removed. To remove all items from the collection use Clear method. The Clear method has no arguments.

## 12.2.5 MonthView Control

The MonthView control is used to display a Calendar that shows one or more months as specified. This control lets the user to move from one month to another. You can also specify selection of dates and the selected dates can be obtained by its value property. One can select one or more dates. However, non-consecutive dates can not be selected.

The MonthView control itself provides a complete interface. The month and year are displayed at the top of the control and two navigation buttons are there, to move to the next or previous month.

By default, the current month of the current year is displayed. The current date is displayed at the foot, encircled in red colour.

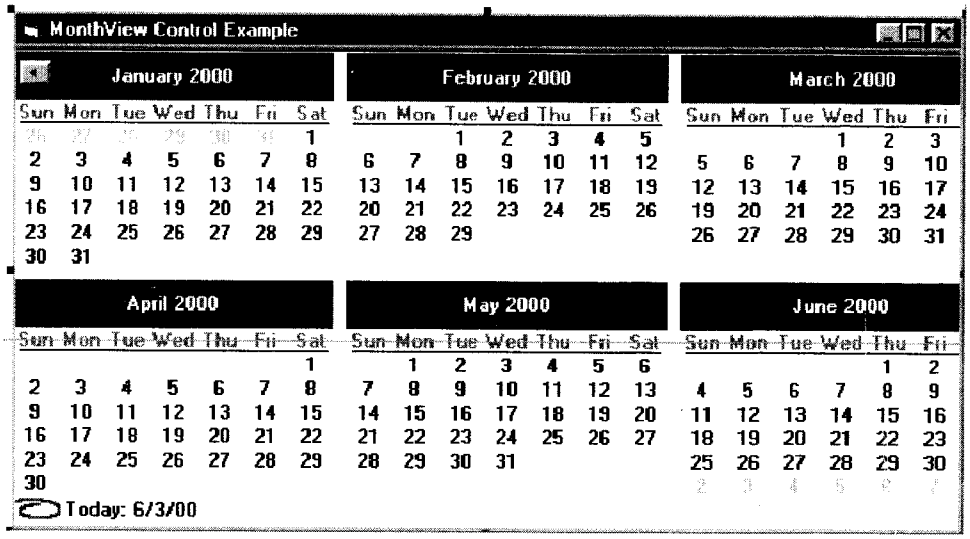



**Figure 12.7**

Now when you are aware with the control, let us proceed to interact with the properties of MonthView control. MonthView control gives you enough flexibility to get a calendar interface of your own choice.

*Month View Control Properties*

- *ForeColor & BackColor*

  The ForeColor property specifies the color of the Date numbers that belong to the current month. BackColor property only changes the thin line at the bottom of the control.

- *TitleBackColor & TitleForeColor*

  The TitleBackColor and the TitleForeColor properties specify the ForeColor and BackColor of the title part of the MonthView Control.

- *TrailingForeColor*

  Specifies the forecolor for the previous or forthcoming months. This makes the current month look different from other months.

- *MonthBackColor*

  Specifies the background color of the month area.

- *MultiSelect*

  MutliSelect property let's you select a date range, if set to True. Single date selection is by default.

- *SelStart, SelEnd*

  SelStart and SelEnd properties reurn the first date and the last date from the date range, selected by user.

- *MonthRows, MonthColumns*

  MonthRows and MonthColumns properties specify the number of rows and columns to be used to display calendar. For example, if you want to display three months down to each other, set the MonthView control's MultiRows property to 3 and MonthColumns property to 1. You can display maximum 12 months at a time, not necessarily of the same year.

- *ShowToday*

  A boolean property that specifies that the current date at the bottom is displayed or not.

- *ShowWeekNumber*

  A boolean property that shows the week number next to each row of dates shown on the monthly calendar. If set to True, weeks are numbered on the basis of first full week of the year. This week is numbered as 1.

- *StartOfWeek*

  Specifies a new starting day of the week.

- *DayBold*

  Property specifies whether or not a particular date is displayed in bold or not. In order to display the selected date in bold use the following code:

  MonthView1.DayBold(MonthView1.Value) = True

  The **VisibleDays** property contains the dates that are displayed on the MonthView calendar. The date in the upper-left corner of the control, (not necessarily to be the first day of the month) corresponds to VisibleDays(1).

*Methods of the MonthView Control*

The MonthView control has only two methods:

- ComputeControlSize
- HitTest

Because the MonthView control gets bigger in size while displaying multiple months, the first method helps you to determine the height and width of the control. The **ComputerControlSize** method fixes the width and height of the MonthView control, even if **MonthRows** and **MonthColumn** properties are set to display multiple months. The syntax of the ComputeControlSize method is:

ComputeControlSize(rows, columns, width, height)

The other method (HitTest) enables you to determine the part of the control where the mouse pointer is positioned. The value returned by this method corresponds to MonthViewHitTest Areas Enum. The syntax for the HitTest method is:

```
VariableName=HitTest(x,y,date)
```

The VariableName is the name of the variable collecting the value returned by HitTest method. X and Y are the mouse pointer's coordinates and the date variable will contain the date that is under the mouse pointer.

Different values returned by MonthViewHitTestAreasEnum are:

| Enum Member | Value |
|---|---|
| mvwCalendarBack | 0 |
| mvwCalendarDate | 1 |
| mvwCalendarDateNext | 2 |
| mvwCalendarDatePrev | 3 |
| mvwCalendarDay | 4 |
| mvwCalendarWeekNum | 5 |
| mvwNoWhere | 6 |
| mvwTitleBack | 7 |
| mvwTitleBtnNext | 8 |
| mvwTitleBtnPrev | 9 |
| mvwTitleMonth | 10 |
| mvwTitleYear | 11 |
| mvwTitleTodayLink | 12 |

*MonthView Control's Important Events*

| | | |
|---|---|---|
| Selchange Event | : | Triggers when the user clicks on a date. |
| DateClick Event | : | Selchange() event is followed by DateClick() event. Occurs only when the user clicks on the date. |
| DateDblClick | : | DateDblClick event is fired when the user double clicks on the date. |
| GetDayBold | : | Occurs whenever the control is initially displayed or the user moves to a previous or later month. |

## 12.2.6 ListView Control

The ListView control is similar to the ListBox, but with enhanced features. This control allows us to display a window that contains a list of information. The ListView control allows us to choose four views to display the list items. Using the ListView control it is possible to organize the list data items into one of the following styles:

| Style | Constants |
|---|---|
| Large Icons | lvwIcons |
| Small Icons | lvwSmall Icons |
| List View | lvwList |
| Report View | lvwReport |

All the items in the ListView control can be accompanied by both icons and text.

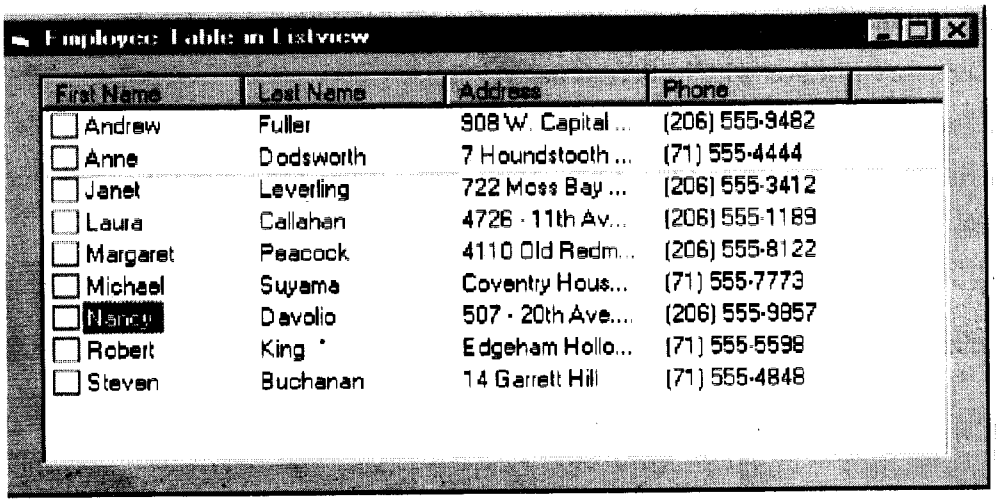



**Figure 12.8**

***Important Properties and Methods***

- ***Arrange:*** Specifies how the icons in the ListView control look like. The three possible values are : lvwNone, lvwAutoLeft, lvwAutoTop.
- ***Icons:*** Returns or sets the index or key for the icon associated with the List item Object
- ***Selected:*** Tells whether the List item is selected.
- ***SmallIcons:*** Returns or sets the index or key for the small icon associated with the List item object.
- ***SubItems:*** Returns or sets the string representing the data for the SubItem associated with a specific List item.
- ***Text:*** The visible text in the control.
- ***View:*** This property sets the display style of the items. It has the following s ettings :

| Settings | Description |
|---|---|
| lvwReport | Displays a simple list |
| lvwSmallIcons | Displays a list using small icons. |
| LvwLargeIcons | Displays a list using large icons |

Important events for the ListView control is the Click() event. Use load event of the form to add all the items in the ListView control.

### *Example Using a ListView Control*

*Explorer Application*

This application fills the ListView control with the directories in the specified path. To create this application follow these steps.

- Add a new standard EXE project
- Add a ListView control with default name.
- Add four command buttons with cmdView, cmdSmall, cmdList and cmdQuit names.
- Add a Directory List box with its default name.
- Add an ImageList control with its default name.
- Now write the following code at respective event handlers after designing the form that looks like the following screen.

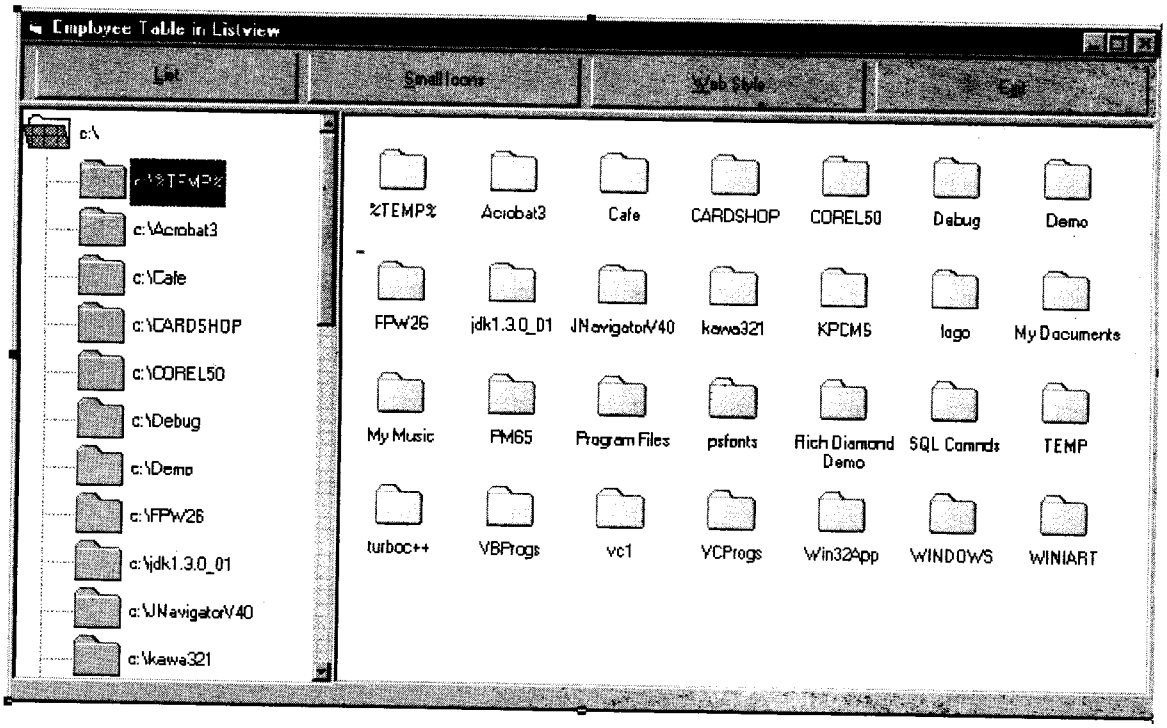



**Figure 12.9**

'Add the following code to declare public variables at the 'general section of the form.

```
Dim cHead As ColumnHeader
    Dim lstImages As ListImage
    Dim lstItem As ListItem
    Dim i As Integer
    Private Sub cmdList_Click()
      Dim lstItem As ListItem
      Dim cHead As ColumnHeader
```

```
    ListView1.ListItems.Clear
    Set cHead = ListView1.ColumnHeaders._
          Add(, ,"",ListView1.Width)
    Set 1stImages = ImageList1.ListImages.Add_
          (, ,LoadPicture("open.bmp"))
    ListView1.Icons = ImageList1
    ListView1.View = lvwList
    ListView1.Arrange = 1
    For i = 0 to Dir1.ListCount -1
          Set 1stItem = ListView1.ListItems.Add(, , Dir1.List(i))
          1stItem.Icon = 1
    Next i
    End Sub
Private Sub cmdQuit_Click()
    Unload me
End Sub
Private Sub cmdSmall_Click()
    ListView1.ListItems.Clear
    ListView1.View = lvwSmallIcon
 Set 1stImages = ImageList1.ListImages.Add_
          (, ,LoadPicture("open.bmp"))
    ListView1.Icons = ImageList1
    ListView1.SmallIcons = ImageList1
    ListView1.Arrange = 1
    For i = 0 to Dir1.ListCount -1
          Set 1stItem = ListView1.ListItems.Add(, , Dir1.List(i))
          1stItem.Icon = 1
          1stItem.SmallIcon = 1
    Next i
End Sub
Private Sub cmdView_Click()
    ListView1.ListItems.Clear
    Set cHead = ListView1.ColumnHeaders_
    Add(, , "Name", ListView1.Width/2)
    Set cHead = ListView1.ColumnHeaders._
    Add(, ,"Type", ListView1.Width/2, lvwColumnCenter)
    ListView1.View = lvwReport
    'Load one image into an ImageList control using the following code.
```

```
        Set lstImages = ImageList1.ListImages.Add_
        (, , LoadPicture("open.bmp"))
        ListView1.Icons = ImageList1
        For i = 0 to Dir1.ListCount -1
              Set lstItem = ListView1.ListItems.Add(, , Dir1.List(i)),
'Dir1.ListIndex))
              lstItem.SubItems(1) = "File Folder"
        Next i
    End Sub
    'change drive at the Load event of the form.
    Private Sub Form_Load()
        ChDrive "c:\"
    End Sub
```

## 12.2.7 TreeView Control

The Tree View control works like outline control. The individual objects are called nodes. The Tree View control has a node's collection that holds information about the nodes in the control. The highest node is given by the Root property.

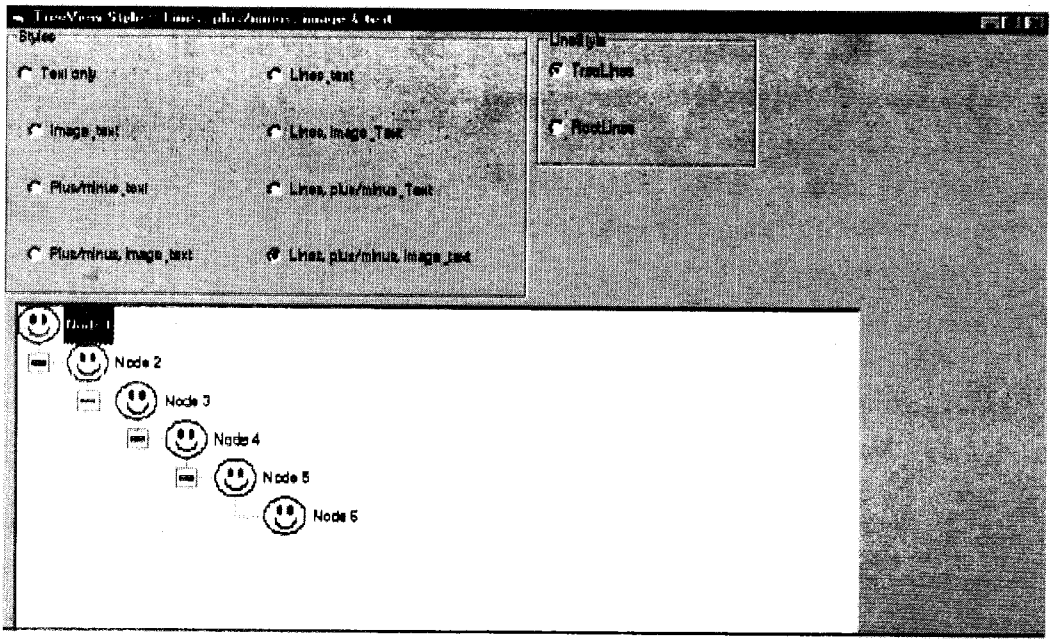



**Figure 12.10**

*TreeView Control's Properties*

Besides regular standard properties of the control, other properties are:

| Values | Description |
|---|---|
| 0-tvwTextOnly | Only the text of the node is displayed |
| 1-tvwPictureText | Shows icons and text both. |
| 2-tvwPlusMinusText | Displays the collapse and expands symbols and the text of the node. |
| 3-tvwPlusPictureText | Displays the collapse/expand symbols, a small icon to the left of the text and the text itself. |
| 4-tvwTreeLinesText | Displays the lines between a node and its hierarchiichal components. Also displays the text. |
| 5-tvwTreeLinesPicture | Displays a small icon to the left of the text, and connects related nodes. |
| 6-tvwTreeLinesPlus | Shows the plus/minus signs, connection lines and MinusText the nodes text. |
| 7-tvwTreeLinesPlus | Shows all the possible settings: connection lines, plus/minus signs, MinusPictureText picture and text. |
| LineStyle | Property determines the style of lines displayed between the nodes.<br>0 - TreeLines<br>1 - RootLines<br>The above are the only possible values for the LineStyle property. |
| LabelEdit | Determines whether or not the text of a node can be edited. A boolean property. Allows editing when set to True. An example of this property can be seen in Internet Explorer's file/folder renaming process (directory or its files). |
| OLEDragMode | Specifies the nature of dragging. Following are the values to this property.<br>0 – OLEDragManual<br>1 – OLEDragAutomatic |
| OLEDropMode | Specifies whether or not drop operations are possible. The values of this property are: |

| Values | Description |
|---|---|
| 0 – OLEDropNone | Does not allow dropping and displays NoDrop icon as a cursor |
| 1 – OLEDropManual | Supports OLEDropping operations handled by the code for drag-drop events. |

| | |
|---|---|
| Indentation | This property determines the horizontal distance between nodes in the Tree View control. Values are numbers. The horizontal spacing increases as the number increases. |
| PathSeparator | You can get or set the delimeter character used for the path. |
| FullPath | Returns the full path of the component selected in the hierarchy of the Tree View control. |
| Scroll | A boolean property specifying whether or not the Scrollbars are displayed in the TreeView control. ScrollBars are visible when number of nodes is too big to fit in the control, if set to True. |
| HotTracking | Displays the full name of the node in a Tool-Tip style when the name of the node does not fit horizontally. |
| Style | Specifies how the control looks and behaves. Different values which can be set for this property are as follows: |

| TreeView | Control's Methods |
|---|---|
| GetVisibleCount | This method returns the number of nodes that are visible in the control without scrolling. |
| HitTest | This method is used to ensure if a desired node is available as a drop target. This method can be useful while handling OLEDragDrop operations. |
| OLEDrag | The OLEDrag method is called to initiate OLEDrag operations. This method triggers the OLEStartDrag event. |
| TheStartLabelEdit | This method turns the labels displaying the text of the nodes intoTextBox. This method can be used even when LabelEdit property is set to False. |

Besides these events you can use Add and Remove methods to add or remove the nodes in the Nodes collection of the TreeView control.

*TreeView Events*

***AfterLabelEdit:*** This event is triggered when the Label editing operations are finished.

***BeforeEdit:*** This event is triggered before the Label editing process starts.

***Collapse and Collapse event:*** Triggers when the user collapses a branch of the tree. Expand This is generally done by double clicking on an expandable node. Expand event triggers when the user expands a branch by double clicking on an expandable node.

***NodeClick:*** NodeClick event procedures accepts the Node as arguments. Any information regarding the node currently selected can be retrieved.

***OLEStartDrag Triggers:*** When a user starts dragging data from the control.

*Occurs:* when OLEDrag Mode property is set to 1.

*OLEDragOver OLEDragOver:* Event is triggered when the OLEDropMode property is set to 1, and OLEData is dropped on the control.

*OLEGiverFeedBack:* OLEDragOver is followed by OLEGiveFeedBack() event.

*Example Using TreeView Control*

To try this example place the following controls on the form.

1. One TreeView control with default name.
2. OptionButton's control array of two elements with default name in a Frame.
3. OptionButton's control array of eight elements with default name in a Frame.
4. One ImageList control with default name and one picture. Choose any or select:
   "C:\ProgramFiles\Microsoft Visual studio\Common\Graphics\Icons\Misc\Face02.ICO"
5. Give reference of ImageList control in ImageList property of TreeView control.

Now add the following code at the Load event of the form.

```
Private Sub Form_Load()
Dim PrintNode As Node
Dim x As Integer
Option1(0).Caption = "Show TreeLines"
Option1(1).Caption = "Show RootLines"
Option2(0).Caption = "Show Text Only"
Option2(1).Caption = "Show Image and Text Both"
Option2(2).Caption = "Show Plus Minus Sign and Text"
Option2(3).Caption = "Show Plus Minus, Image and Text"
Option2(4).Caption = "Show Lines and Text"
Option2(5).Caption = "Show Lines, Image and Text"
Option2(6).Caption = "Show Lines, Plus, Minus and Text"
Option2(7).Caption = "Show Lines, Plus, Minus, Image and Text"
Option1(0).Value = True
Option2(1).Value = True
TreeView1.Style = tvwText
SetPrintNode = TreeView1.Nodes.Add(, , ,"Parent Node",1)
For x = 1 to 10
Set PrintNode = TreeView1.Nodes.Add(x,tvwChild,,"Child_Node" & str(x+1),1)
Next x
PrintNode.EnsureVisible
End Sub
```

Add the following code to decide which LineStyle to display.

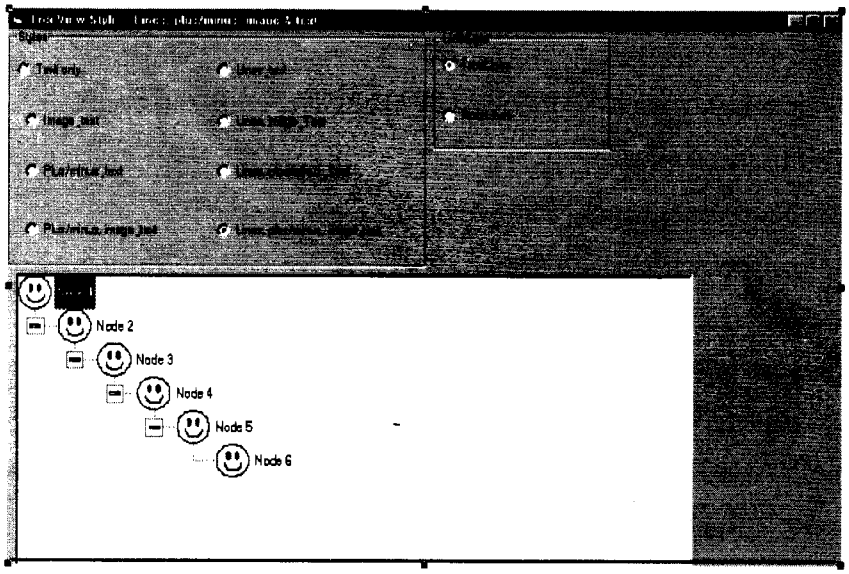



**Figure 12.11**

```
Private Sub Option1_Click(Index As Integer)
TreeView1.LineStyle = Index
End Sub
Private Sub Option2_Click(Index As Integer)
TreeView1.Style = Index
Form1.Caption = "Tree View Style" = "& Option2(Index).Caption"
End Sub
```

This example shows different Styles and LineStyles that can be used with Tree View Control.

## 12.2.8 Microsoft Masked Edit Controls

Microsoft masked edit control is used for input validations in a text box. It looks like an ordinary text box. You can restrict the characters entered without having to write code in the key events. The most important property while working with the Mask control is Mask property. This property controls what the user can enter. Settings for this property are:

| | |
|---|---|
| # | Allows the user to enter a digit only |
| . | Decimal placeholder |
| , | Thousand separator |
| : | Time separator |
| / | Date separator |
| ? | Letter Placeholder A-Z or a-z. |
| & | Allows the user to enter a single ANSI character between 32-126 and 128-256. |
| > | Converts all characters that follow to uppercase. |

< Converts all characters that follow to lowercase.

A Requires an Alphanumeric character.

a Allows an Alphanumeric character.

9 Allows a digit to be entered.

Example : Mask1.Mask = ("###-##-###")

The only event in dealing with input to Masked Edit box is the ValidationError event. This event is triggered whenever a user tries to enter an invalid character.

## 12.2.9 FlatScrollBar Control

You have already worked with the Scrollbars in the previous chapter. The FlatScrollBar is nothing different from the old vertical and horizontal Scrollbars. FlatScrollbar is more flexible and interactive. FlatScrollBar changes the appearance when the mouse cursor is positioned over them, and even can change their color. FlatScrollBars have different modes of appearance and reacts accordingly on user's actions. A FlatScrollBar can be seen in Internet Explorer.

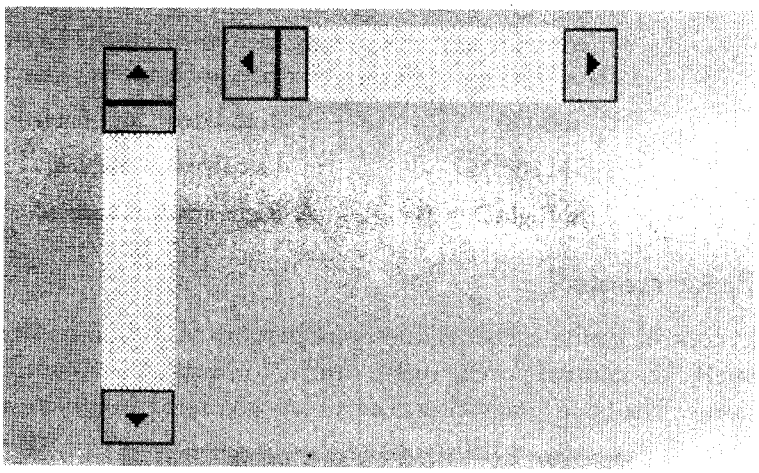


Figure 12.12

FlatScrollbars have generally two modes, the default 2D mode and the beveled 3D mode. 3D mode triggers when the FlatScrollbars are under the mouse pointer. The FlatScrollbars look like common Horizontal and Vertical scrollbars. Scroll arrows of the FlatScrollbars can be disabled in order to give a Visual clue to the users. Gradually FlatScrollbars are taking over the Horizontal and Vertical Scrollbars and even the newer versions of Microsoft's applications are using FlatScrollbars because of their flexibility.

*Properties*

The FlatScrollbars being identical to HScrollbars and VScrollBars possesses most of the properties like Max, Min, LargeChange, SmallChange, Value. These properties have already been discussed earlier. FlatScrollbars also have the maximum and minimum limit to scroll that can be set using Max and Min properties. Value property returns the current value of the FlatScrollBars which can be read or set both at any time. Besides properties common to HScrollbar and VScrollbars other properties are :

*Appearance:* Specifies the way in which control appears and reacts to user's actions. Value that can be set for this property are:

| Value | Description |
|---|---|
| 1 (fsbFlat) | This is default value and gives a FlatScrollBar a 2D appearance.<br>The thumb of the FlatScrollBar and Scroll arrows change the color when the mouse pointer is positioned over them. |
| 2 (fsbTrack3D) | If the Appearance property of the FlatScrollBars is set to this value, they retain their 2D appearance but the thumb and the scroll arrows become 3D when the mouse cursor points at them. |
| 0 (fsb3D) | This value makes the FlatScrollBar look like older HScrollBars and VScrollBars. |
| Orientation | The Orientation property lets you specify whether the Scrollbar will be Horizontal or Vertical. Default is the Horizontal configuration. The values that can be set for this property are:<br>0 (SldHorizontal) Horizontal Orientation<br>1 (SldVertical) Vertical Orientation |
| Arrows | Arrows property specifies the arrows which will be enabled. By default both the arrows of the FlatScrollbar are enabled. The values for this property are:<br>0 (fsbBoth) Both arrows are enabled.<br>1 (fsbLeftUp) Left arrow is enabled<br>2 (fsbRightDown) Right arrow is enabled. |

## 12.2.10 DateTimePicker Control

The DateTimePicker control canbe called as the combination of a ComboBox and a MonthView control. The DateTimePicker control is very much similar to MonthView control but however these two controls are not same. The DateTimePicker control lets you select both date and time.

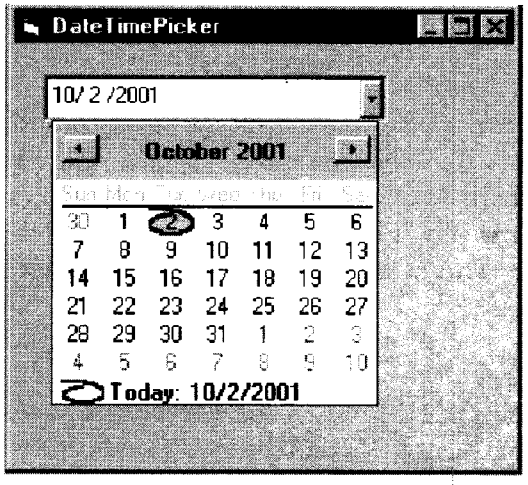



Figure 12.13

The DateTimePicker control provides four different formats for values.

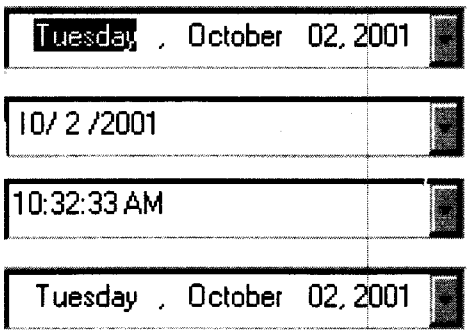



**Figure 12.14**

The DateTimePicker control displays a drop down calendar like the MonthView control, but does not provide enough properties to control the display settings of the control. This control can be very useful and may provide a powerful interface in application that needs date, time inputs from the user. Now lets explore the properties, methods and events of this control.

*Properties of the DtPicker Control*

Although DateTimePicker or say DTPicker control has very few properties that let you control the display settings of the calendar, most of them resemble the MonthView control's properties.

- *Colour Properties*

  The CalendarBackColor, CalendarForeColor controls the back colour and the colour of the dates of the calender and resembles the MonthBackColor and ForeColor properties respectively. The CalendarTrailingForeColor specifies the colour of dates from previous or trailing months.

  The CalendarTitleForeColor and CalendarTitleBackColor specifies the fore colours and back colours of the TitleBar that shows the month and year.

- *MinDate and MaxDate*

  MinDate and MaxDate properties control the range of date that can be displayed, similar to the MonthView control.

- *Value*

  The Value property returns the date or time selected. Similarly, Month, Year, Day, Hour, Minute and Second properties return specific information from the DateTimePicker control's value property.

- *DayOfWeek*

  Starting day of the week can be specified by DayOfWeek property, instead of Sunday which is default. This property is similar to StartOfWeek property used in the MonthView control.

- *UpDown*

  Up and Down arrow keys can be used for exploring in the Calendar, if the UpDown Property is set to true.

- *CheckBox*

  The CheckBox property when set to True displays a checkbox that shows whether any date is selected or not. If no date is selected the checkbox is unmarked and marked when the date is selected.

- *Format*

  This property is used to determine the format of the date and time to be displayed in the DateTimePicker control. Format property has the following syntax

  Object.Format = Integer

  The settings for integer values are:

| Constant | Value | Example |
|---|---|---|
| DtpLongDate | 0 | "Monday, May 17 1979" |
| DtpShortDate | 1 | "11/12/76" |
| DtpTime | 2 | "10:20:55" |
| DtpCustom | 3 | Custom format specified in the CustomFormat property. |

- *CustomFormat*

  Returns or sets the value that specifies the format used by the control when displaying date or time in the box. CustomFormat property works only when the value of the Format property is set to DtpCustom.

  Specifications in the CustomFormat property includes format characters which are given below. The property values are case sensitive.

| Format Characters | Description |
|---|---|
| d | One-or-Two-digit Day. |
| dd | Two digit day |
| ddd | Week Day abbreviation of three characters e.g., MON,TUE etc. |
| dddd | Full Week Day name |
| h | One-or-Two Digit hour in 12 hour format, e.g., 6,1,11 etc. |
| hh | Two digit hour in 12 hour format, e.g., 09,12,06 etc. |
| H | One-or-Two Digit hour in 24 hours format. |
| m | One-or-Two Digit minute |
| mm | The Two-digit minute. |
| M | One-or-Two digit month number. |
| MM | Always two digit month number, e.g., 01,02,07 |
| MMM | Month abbreviation of three characters. |
| MMMM | Full month name. |

| | |
|---|---|
| S | One-or-two digit Second. |
| SS | Always two digit second. |
| t | One letter AM/PM abbreviation. |
| tt | Two-letter AM/PM abbreviation. |

X Callback field. The control uses the other valid format characters and asks the user to fill in the "X" portion. Multiple "X" characters can be used in series to signify unique callback fields.

| | |
|---|---|
| y | One-digit year. |
| YY | Last Two digits of the year. |
| yyy | Full year with the century. |

As discussed above, the DTPicker control uses the other valid format characters and asks the user to fill in the "X" portion, called callback field. In order to use this callback field you must set the Format property to DtpCustom and CustomFormat property with a new string containing format.

For example, you want the date to be displayed in the following format, where the month names are in Spanish language.

Julio, Saturday, 1976

The first step is to set the format property to DtpCustom and CustomFormat property with the following format

xxxx,dddd,yyy

The "x" included here determines the Custom name for the month.

- ***Events of DtPicker Control***

Everytime when the new value is asked by the user, FormatSize event is fired which has callbackField as string and size as integer arguments. This event allows you tos et the maximum allowable size of the formatted string.

- ***Dim NewMonthString(12) As String***

At the Load event of form write the following code. Declare the NewMonthString array at the general section of the form.

```
NewMonthString(0) = "Enero"
NewMonthString(1) = "Febrero"
NewMonthString(2) = "Margo"
NewMonthString(3) = "April"
NewMonthString(4) = "Mayo"
NewMonthString(5) = "Julio"
NewMonthString(6) = "Jimio"
NewMonthString(7) = "Agosto"
NewMonthString(8) = "Septiembre"
```

```
NewMonthString(9) = "Octbre"
NewMonthString(10) = "Noviembre"
NewMonthString(11) = "Diciembre"
```

This code will store all the Spanish names of the months to an array NewMonthString. To set the size of the format write the following code at FormatSize Event Procedure.

```
Private Sub DtPicker_FormatSize(ByVal CalbackField as String, Size As
Integer)
Dim MonthLen As Integer
If callbackfield = "xxxx" then
MonthLen = 0
For x = 0 to 11
If MonthLen < Len(NewMonthString(x)) then
MonthLen = Len(NewMonthString(x))
End if
Next i
End if
Size = MonthLen
End Sub
```

The above code fixes the size argument with new value which is the length of new string containing month. The value of size may vary depending upon the length of the month name. You have already declared one string type array of twelve elements in the general declaration section of your form module with the name NewMonthString.

Now the only code you need to write before running your application is for setting the new value of the month name. This can be done by writing the following code at Format event. The Format event has callbackField argument as string, FormattedString as string.

```
Private Sub DtPicker_Format(ByVal CallbackField as String, FormattedString
as String)
If callbackField = "xxxx" then
FormattedString = NewMonthString(DtPicker1.Month)
End if
End Sub
```

This code will store the currently selected month which is an integer returned by DtPicker1.Month.

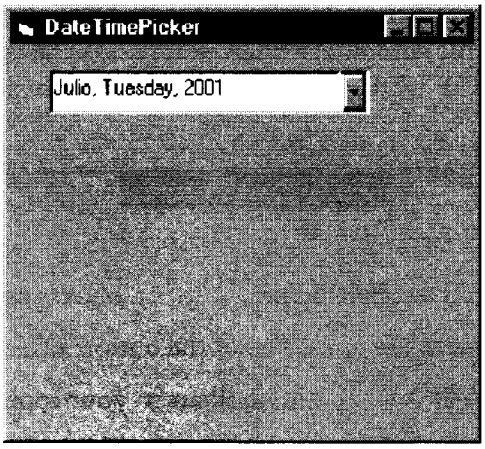



**Figure 12.15**

## 12.3 WINSOCK CONTROL

A WinSock control allows you to connect to a remote machine and exchange data using either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). Both protocols can be used to create client and server applications. Like the Timer control, the WinSock control doesn't have a visible interface at run time.

*Possible Uses*

- Create a client application that collects user information before sending it to a central server.
- Create a server application that functions as a central collection point for data from several users.
- Create a "chat" application.

*Selecting a Protocol*

When using the WinSock control, the first consideration is whether to use the TCP or the UDP protocol. The major difference between the two lies in their connection state:

- The TCP protocol control is a connection-based protocol, and is analogous to a telephone — the user must establish a connection before proceeding.
- The UDP protocol is a connectionless protocol, and the transaction between two computers is like passing a note: a message is sent from one computer to another, but there is no explicit connection between the two. Additionally, the maximum data size of individual sends is determined by the network.

The nature of the application you are creating will generally determine which protocol you select. Here are a few questions that may help you select the appropriate protocol:

1. Will the application require acknowledgment from the server or client when data is sent or received? If so, the TCP protocol requires an explicit connection before sending or receiving data.
2. Will the data be extremely large (such as image or sound files)? Once a connection has been made, the TCP protocol maintains the connection and ensures the integrity of the data. This connection, however, uses more computing resources, making it more "expensive."

3. Will the data be sent intermittently, or in one session? For example, if you are creating an application that notifies specific computers when certain tasks have completed, the UDP protocol may be more appropriate. The UDP protocol is also more suited for sending small amounts of data.

**Check Your Progress**

Fill in the blanks:

1. A toolbar control contains a collection of ..... objects used to create a toolbar.
2. The index parameter returns the ..... of the button added.
3. Coolbar enables you to add ..... toolbars to your applications.
4. The DateTimePicker control displays a ..... calendar like the MonthView control.

## 12.4 LET US SUM UP

ActiveX is a set of reusable Components that can be created and utilized by several applications. ActiveX uses the Internet technology to assist in creating compact and reusable applications that can be deployed via the Internet or a corporate Intranet.

ActiveX controls are the custom controls that can be added to the ToolBox and used in several VB applications.

While working with Toolbar or Coolbar controls you need to assign an image to each button. Instead of loading each image into the memory before using, it is easier to store the reference of all images in a single place.

A toolbar control contains a collection of Buttons objects used to create a toolbar.

The new control introduced by Microsoft to replace a toolbar control is a Coolbar control. Coolbar enables you to add sliding toolbars to your applications.

The MonthView control is used to display a Calendar that shows one or more months as specified.

## 12.5 KEYWORDS

***ActiveX:*** It is a set of reusable Components that can be created and utilized by several applications.

***ActiveX Controls:*** These are the custom controls that can be added to the ToolBox and used in several VB applications.

***ImageList:*** This property is used to specify the name of the control containing images, which can be displayed next to the band's move handles.

***Indentation:*** This property can be used to specify a default indentation for any ComboItems object, belonging to ComboItem's collection.

## 12.6 QUESTIONS FOR DISCUSSION

1. State whether the following statements are true or false:
   (a) ActiveX component are reusable components.
   (b) An ActiveX control cannot be created using VB.
   (c) A Coolbar control can be slid in its pane.
   (d) A DateTimePicker control displays only data.
   (e) A Bands collection is a collection of Band object found in the Toolbar control.
   (f) An ImageList can be instantiated at run time.
   (g) The following statement:

   ImageCombo1.Add(1,"A","FirstItem",ImageList1.ListImage(1),_ ImageLis1.ListImages(2),1) adds a new item to an ImageCombo control with images to be displayed against the item and indentation level is 1.

2. Match the following:

| | |
|---|---|
| (i) Toolbar | a. MonthView Control |
| (ii) Coolbar | b. ImageCombo |
| (iii) ImageList | c. DateTimePicker |
| (iv) Indentation | d. ListView |
| (v) MinHeight | e. Bands |
| (vi) TrailingForeColor | f. TreeView |
| (vii) Validation error event | g. Coolbar |
| (viii) Displays a window containing list of information | h. Buttons |
| (ix) Contains a list where each item is referred as node. | i. ListImages |
| (x) Lets you select both date and time | j. Masked Edit Control. |

3. Write two ways of adding images in an ImageList control with example.
4. Explain the advantage of Coolbar control over Toolbar control.
5. Differentiate between a ListView control and a TreeView control.

**Check Your Progress: Model Answers**

1. Buttons
2. Index
3. Sliding
4. drop down

## 12.7 SUGGESTED READINGS

*SAMS Teach Yourself Visual Basic 6 in 24 Hours;* Greg M. Perry, Sanjaya Hettihewa; SAMS Publishing; 1988

*Visual Basic 2005 in a Nutshell*; Paul Lomax, Steven Roman, Ron Petrusha, Tim Patrick; O'Reilly; 2006

*Beginning Visual Basic 6*; Peter Wright; Wrox Press; 1998

*Visual Basic Complete;* Steven Brown; Sybex; 1999

*Visual Basic 6 for Dummies*; Wallace Wang; For Dummies; 1998