PDCA - 04

VARDHAMAN MAHAVEER OPEN UNIVERSITY, KOTA

Post Graduate Diploma in Computer Application
(PGDCA)

VB and DOT NET Technology

Production July 2007

INDEX

Unit - 1: Business Data Processing

Structure of the Unit

1.0 Objectives

At the end of this unit, you will be able to,

- Understand file organization, concept of records and fields
- Distinguish Sequential and Random files
- Understand concept of an organization
- Appreciate the need for Visual programming
- Explain the Visual Basic and its features

1.1 Introduction

In a modern computer based information system files play an important role. A file is a collection of records. Each record is made up of fields. Groups of fields are combined to form a logical record. This logical record contains all the data of interest about some entity.

There are various kinds of file structures. The simplest kind of file is the one in which all the records are in sequence. Many computer applications rely on sequential files. Another kind of organization is direct or random access file organization. In this case, records are stored without regard to the sequence of record keys. The location of a record is determined by transferring record key into a physical address.

1.2 File Organization

A file is a collection of data that the system maintains in persistent storage. Persistent means that the storage is non-volatile - that is, the system maintains the data even after the program terminates; indeed, even if you shut off system power. For this reason, plus the fact that different programs can access the data in a file, applications typically use files to maintain data across executions of the application and to share data with other applications.

Files generally take one of two different forms: *sequential files* or *random access files*. Sequential files are great for data you read or write all at once; random access files work best for data you read and write in pieces (or rewrite, as the case may be). For example, a typical text file is usually a sequential file. Usually your text editor will read or write the entire file at once. Similarly, the HLA compiler will read the data from the file in a sequential fashion without skipping around in the file. A database file, on the other hand, requires random access since the application can read data from anywhere in the file in response to a query.

1.3 RECORDS AND FIELDS

Records the building blocks of databases, records describe information sources.

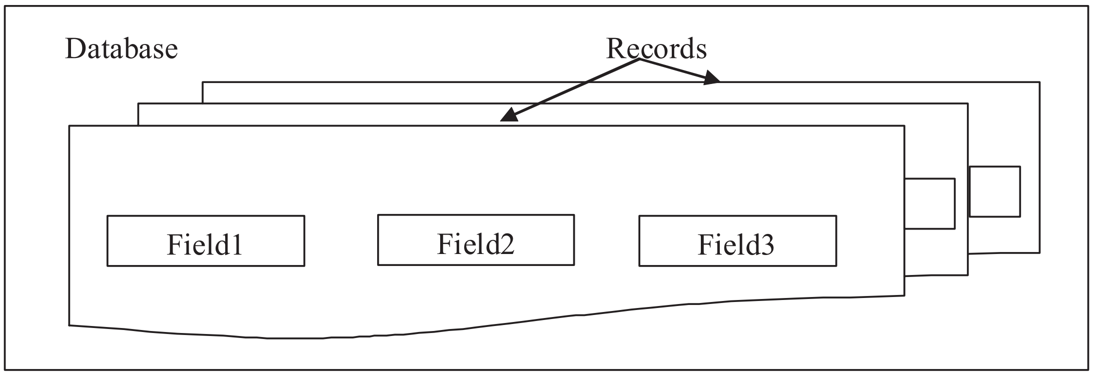


Figure 1: Database, records and fields

A good view of a file is as a list of records. That is, the file is a sequential string of records that share a common structure. A list is simply an open-ended single dimensional array of items, so we can view a file as an array of records. As such, we can index into the file and select record number zero, record number one, record number two, etc. Using common file access operations, it is possible to skip to different records in a file.

Fields are the building blocks of records; these are the sections of a record where information is stored.

For example, your driver's license or ID card is a record about you with fields describing your name, eye color, height, address, and so on.

Depending on the database designer's choice, field can vary. In a library database, for example, author, title, subject, publisher, and publication date are the common fields.

1.4 Master and Transaction DATA

When data processing takes place, generally two types of data are used, transaction data and master data. Transaction data are used to update the master data. The master data is the more permanent data, which is kept updated by the transaction data.

1.4.1 Master Data

Master data is the data without which you cannot do any transactions and is mandatory for every organization. It describes the things that interact when a transaction occurs. For instance, master data that represents product and customer must be present before the transaction is fired to sell a product to a customer.

Characteristics of Master Data:

- First, the need for a master copy indicates that there may be copies of the same or similar data objects used in contexts where a lack of synchrony between copies leads to inconsistency across applications dependent on those copies.
- Second, the desire to subject the data to management indicates a willingness of the stakeholders to collaborate on centralized governance over the master copies.
- Third, master data objects are both the subject of transactions (as part of operational systems) and analysis (as part of analytical systems).
- Fourth, the concept of "master" implies that all application uses are subsidiary to a single core repository.
- Fifth, a master object can be assigned a unique identifier within the enterprise.

Master objects are considered to be subjects of transactions or analysis. However, transactions themselves are subject to transactions and analysis; transactions may be composed into workflows, which, in turn, are also subject to transactions and analysis. Therefore, transactions could be represented as master objects, allocated a unique system identifier, and then be subjected to various productivity and performance analyses.

Examples of transaction data are, employee details, customer details, product details, student details etc.

1.4.2 Transaction data

These are the business documents that are created using the master data, like purchase orders, sales orders etc. Transactional Data can change very often and are not constant. And this data is created/modified out of an application transaction.

Examples of transaction data are, customers' orders for product, details of price changes for product, details of cash posting in customer account, records of items sold to customer etc.

1.5 Sequential and Random File Access

Sequential access refers to reading or writing data records in sequential order, that is, one record after the other. To read record 100, for example, you would first need to read records 1 through 99. In random access, you can read and write records in any order. To read record 100, in random access, you can jump directly to record 100. The terms random access and sequential access are often used to describe data files.

1.5.1 Sequential Files

In a Sequential file the records are arranged serially, one after another. The only way to access records in a Sequential file, is serially. A program must start at the first record and read all the succeeding records until the required record is found or until the end of the file is reached.

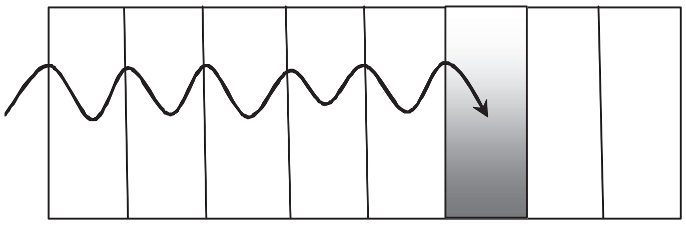


Figure 2: Sequential Access File

Sequential files may be *Ordered* or *Unordered* (these should be called Serial files). The ordering of the records in a file has a significant impact on the way in which it is processed and the processing that can be done on it.

In an ordered file, the records are sequenced on some field in the record (like StudentId or StudentName etc). In an unordered file, the records are not in any particular order.

Ordered File	Unordered File
RecordA	RecordM
RecordB	RecordH
RecordG	RecordB
RecordH	RecordN
RecordK	RecordA
RecordM	RecordK
RecordN	RecordG

Advantages of Sequential Files

1. Very easy to process,
2. Can be easily shared with other applications developed using different programming languages.

Disadvantages of Sequential Files

Sequential files can be only processed sequentially. If you need to read record number N, you must first read the previous N-1 records. Especially not good for programs, that make frequent searches in the file.

Records in an ordered Sequential file are arranged, in order, on some key field or fields. When we want to insert, delete or amend a record we must preserve the ordering. The only way to do this is to create a new file. In the case of an insertion or update, the new file will contain the inserted or updated record. In the case of a deletion, the deleted record will be missing from the new file.

The main drawback to inserting, deleting or amending records in an ordered Sequential file is that the entire file must be read and then the records written to a new file. Since disk access is one of the slowest things we can do in computing this is very wasteful of computer time when only a few records are involved.

For instance, if 10 records are to be inserted into a 10,000 record file, then 10,000 records will have to be read from the old file and 10,010 written to the new file. The average time to insert a new record will thus be very great. To solve this problem random files are introduced.

1.5.2 Random (Direct) Access Files

Direct access files allow direct access to a particular record in the file using a key and this greatly facilitates the operations of reading, deleting, updating and inserting records. Direct files are also known as relative files.

Records in relative files are organized on ascending Relative Record Number. A Relative file may be visualized as a one dimension table stored on disk, where the Relative Record Number is the index into the table. Relative files support sequential access by allowing the active records to be read one after another.

Relative files support only one key. The key must be numeric and must take a value between 1 and the current highest Relative Record Number. Enough room is allocated to the file to contain records with Relative Record Numbers between 1 and the highest record number.

For instance, if the highest relative record number used is 10,000 then room for 10,000 records is allocated to the file.

Figure 1 below contains a schematic representation of a Relative file. In this example, enough room has been allocated on disk for 328 records. But although there is room for 328 records in the current allocation, not all the record locations contain records. The record areas labeled "free", have not yet had record values written to them.

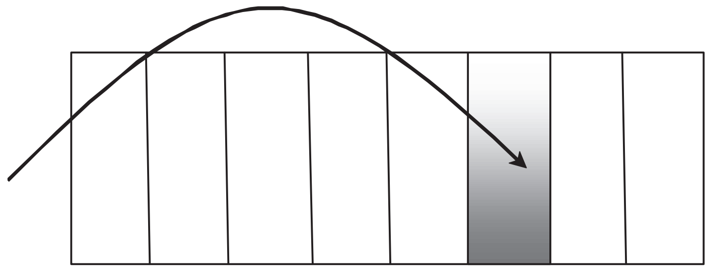


Figure 3: Random Access File

Advantages of Random Access Files

1. In comparison to sequential access files, you may significantly save on the amount of disk space required by the file by using random access.
2. With random access, files can be opened for both read/write at the same time. This is a great advantage over using sequential access.

Disadvantages of Random Access Files

You can have wasted space if many fields in the records are left blank or if most of the strings stored in the record are significantly shorter than the length dimensioned for the string.

Self Learning Exercise-1

i. Fill in the blanks

a) Direct files are also known as ....

b) Files generally take one of two different forms:…………….or …………

c) A record consists of several………

ii. State True or False

a) The advantage of sequential file is that it needs no sorting.

b) Every record in a sequential file must process a key field.

c) Records in relative files are organized on ascending Relative Record Number.

d) File is a collection of fields.

1.6 Concept of Organization

An organization is a group of members intentionally organized to accomplish an overall, common goal or set of goals.

1.6.1 Features of Organization

Vision

Members of an organization often have some image in their minds about how the organization should be working, how it should appear when things are going well.

Mission

An organization operates according to an overall purpose, or mission.

Values

All organizations operate according to overall values, or priorities in the nature of how they carry out their activities. These values are the personality, or culture, of the organization.

Strategic Goals

Members of an organization often work to achieve several overall accomplishments, or goals, as they work toward their mission.

Strategies

Organizations usually follow several overall general approaches to reach their goals.

Systems and Processes that (Hopefully) are Aligned with Achieving the Goals

Organizations have major subsystems, such as departments, programs, divisions, teams, etc. Each of these subsystems has a way of doing things to, along with other subsystems, achieve the overall goals of the organization. Often, these systems and processes are defined by plans, policies and procedures.

1.6.2 Organization as a System

It helps to think of organizations as systems. Simply put, a system is an organized collection of parts that are highly integrated in order to accomplish an overall goal. A system has various inputs which are processed to produce certain outputs, that together, accomplish the overall goal desired by the organization. There is ongoing feedback among these various parts to ensure they remain aligned to accomplish the overall goal of the organization. There are several classes of systems, ranging from very simple frameworks all the way to social systems, which are the most complex. Organizations are, of course, social systems.

Systems have inputs, processes, outputs and outcomes. To explain, *inputs* to the system include resources such as raw materials, money, technologies and people. These inputs go through a *process* where they're aligned, moved along and carefully coordinated, ultimately to achieve the goals set for the system. *Outputs* are tangible results produced by processes in the system, such as products or services for consumers. Another kind of result is *outcomes*, or benefits for consumers, e.g., jobs for workers, enhanced quality of life for customers, etc. Systems can be the entire organization, or its departments, groups, processes, etc.

Feedback comes from, e.g., employees who carry out processes in the organization, customers/clients using the products and services, etc. Feedback also comes from the larger environment of the organization, e.g., influences from government, society, economics, and technologies.

Each organization has numerous subsystems as well. Each subsystem has its own boundaries of sorts, and includes various inputs, processes, outputs and outcomes geared to accomplish an overall goal for the subsystem. Common examples of subsystems are departments, programs, projects, teams, processes to produce products or services, etc. Organizations are made up of people — who are also systems of systems of systems — and on it goes. Subsystems are organized in an hierarchy needed to accomplish the overall goal of the overall system.

The organizational system is defined by, e.g., its legal documents (articles of incorporation, by laws, roles of officers, etc.), mission, goals and strategies, policies and procedures, operating manuals, eta. The organization is depicted by its organizational charts, job descriptions, marketing materials, etc. The organizational system is also maintained or controlled by policies and procedures, budgets, information management systems, quality management systems, performance review systems, etc.

1.7 Different Management Levels

There are three different levels of management:

1. Top level management, consists of Board of directors, managing directors or executive committee members.
2. Middle level management, consists of managers such as personnel, production, sales, marketing, finance, operations etc.
3. Lower or operating level management, consists of foreman, supervisors, daily laborers etc.

1.7.1 Top Level Management

Top level management refers to a group of individuals who occupy functional positions in an enterprise such as Board of directors, general managers and other key officers who are responsible for smooth and systematic operations of the enterprise. Top management does not directly execute work. Experts say, it is a concept of functions concerning the manner in which the enterprise should achieve its goal. Generally for a large enterprise key functions and duties cannot be carried out by individuals, hence a compact group of members is formed. Top level management should focus more on capability of workers both in general and technical qualities. It involves creative imagination, initiative, and sense of judgments. It is also described as policymaking group responsible for overall direction and activities of the enterprise.

Main objective in Top level management are:

1. Setting key objectives, policies and identifying factors essential for the development of the enterprise.
2. Efficient accomplishment of goals in the enterprise and maintaining strategic balance in all actions taken by the authorities in higher level.
3. Making appointments to the top position in the enterprise such as managers, department heads etc.
4. Reviewing the work of different personnel in all levels.

The function of the top level management is providing a detailed description of the nature of their activities in the objects clause of their memorandum of association. Objectives may also be specific such as specialty in workmanship, competitive pricing, marketing, and relationship with the customers, workers, and public. Framing up policy such as production policy indicates schedule of productions to be completed. Product policy lays down size, color, material, design etc. Marketing policy focuses on various channels of selling. A personnel policy deals with recruitment and placement. It includes organizing which deals with allocation of duties to the personnel. Controlling plays a key role in the top level management, which makes comparisons with the actual results and planned targets.

1.7.2 Middle level management

Middle level management deals with the task of implementing the policies and plans formulated by the top level. It comprises of departmental heads and other executive officers who will lead the group of workers to the planned targets and provide them with necessary resources in order to get the job done. This group is responsible for the execution and interpretation of policies throughout the organization and for the successful operations assigned to the division or departments. In this level the managers have to plan the operations, issue instructions laid by the top management, collect the resources required and control the work of the men. Managers are responsible for leading all the function within each department; they provide the guidance and structure for a purposeful enterprise. Functions to be performed in the middle level management are.

1. Follow the rules and policies formulated by the top management.
2. Motivating personnel for higher productivity.
3. Collecting detailed analysis report of the department and the personnel.
4. Mutual understanding with other departments in the enterprise.
5. Recommendations to top management.

1.7.3 Low level management or Operative management

It is the lowest level in the business enterprise Foreman, supervisor executives assisted by number of workers carry out the process to be done as per schedule. Their authority and responsibility in the enterprise will be very much less compared to other workers. They have to follow the rules and guidelines made out by the higher authorities of the enterprise. The importance of the functions in this level cannot be overlooked. The plan developed by the top level management will fail if the workers in the lower level do not fully realize the work allotted to them and the nature of their work.

The quality and quantity of the work done will depend upon the performances of the workers in this level how hard they work to attain their goals. The supervisors in this level have to maintain standards of the quality of the manufactured product, assign duties to the workers as per plan and schedules given by the top and middle level management. They are also responsible for maintaining respect, discipline among themselves and increase the spirit of work among the workers.

Self Learning Exercise - 2

i. Fill in the blanks

d)……… management deals with task of implementing the policies and plans formulated by the top level.

e) Low level management is known as ....

ii. State True or False

e) Organization is not a system.

f) Systems have inputs, processes, outputs and outcomes.

1.8 Introduction to Visual Basic

Visual Basic is a programming language and environment developed by Microsoft. Based on the BASIC language, Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces. Instead of worrying about syntax details, the Visual Basic programmer can add a substantial amount of code simply by dragging and dropping controls, such as buttons and dialog boxes, and then defining their appearance and behavior.

Although not a true object-oriented programming language in the strictest sense, Visual Basic nevertheless has an object-oriented philosophy. It is categorized as an object based programming language as it does not support all features of an OOP language. It is also called an event-driven language because each Visual Basic object can react to different events such as a mouse click.

Since its launch in 1990, the Visual Basic approach has become the norm for programming languages. Now there are visual environments for many programming languages, including C, C++, Pascal, and Java. Visual Basic is sometimes called a Rapid Application Development (RAD) system because it enables programmers to quickly build prototype applications.

Visual Basic

- **Visual**: Refers to the method used to create GUI. Rather than writing many lines of code for the appearance & location of interface elements used in a program, we can simply add pre-built objects on the screen.
- **BASIC**
  - Beginner's All-purpose Symbolic Instruction Code
  - By John Kemeny and Thomas Kurtz
  - Designed to teach programming to beginners
  - BASIC language interpreter, one of Microsoft's first products
- **BASIC**
  - Included with DOS

    - QBASIC - included with DOS Version 5 - subset of BASIC
- **Visual Basic**
    - 1992 - Visual Basic 1 for Windows 3
    - Visual Basic 4 - for Windows 95
- **Visual Basic 5 - for Office 97**
    - Visual Basic 6 - for Windows 98 & Office 2000
    - Visual BASIC .NET

1.9 Features of Visual Basic

Visual Basic was designed to be easy to learn and use. The language not only allows programmers to easily create simple GUI applications, but also has the flexibility to develop fairly complex applications as well. Programming in VB is a combination of visually arranging components or controls on a form, specifying attributes and actions of those components, and writing additional lines of code for more functionality. Since default attributes and actions are defined for the components, a simple program can be created without the programmer having to write many lines of code.

Forms are created using drag and drop techniques. A tool is used to place controls (e.g., text boxes, buttons, etc.) on the form (window). Controls have attributes and event handlers associated with them. Default values are provided when the control is created, but may be changed by the programmer. Many attribute values can be modified during run time based on user actions or changes in the environment, providing a dynamic application. For example, code can be inserted into the form resize event handler to reposition a control so that it remains centered on the form, expands to fill up the form, etc. By inserting code into the event handler for a keypress in a text box, the program can automatically translate the case of the text being entered, or even prevent certain characters from being inserted.

Visual Basic can create executables (EXE), ActiveX controls, DLL files, but is primarily used to develop Windows applications and to interface web database systems. Dialog boxes with less functionality (e.g., no maximize/minimize control) can be used to provide pop-up capabilities. Controls provide the basic functionality of the application, while programmers can insert additional logic within the appropriate event handlers. For example, a drop-down combination box will automatically display its list and allow the user to select any element. An event handler is called when an item is selected, which can then execute additional code created by the programmer to perform some action based on which element was selected, such as populating a related list.

The language is garbage collected using reference counting, has a large library of utility objects, and has basic object oriented support. Since the more common components are included in the default project template, the programmer seldom needs to specify additional libraries. Unlike many other programming languages, Visual Basic is generally not case sensitive, although it will transform keywords into a standard case configuration and force the case of variable names to conform to the case of the entry within the symbol table entry. String comparisons are case sensitive by default, but can be made case insensitive if so desired.

1.10 Need of Visual Basic

VB is a visual programming language. Before the invention of visual programming, a programmer spends a lot of time writing code for the user interface. Moreover, a large percentage of the time is wasted in actions such as aligning the buttons properly, determining which button has been clicked and then executing the appropriate code.

The functionality of each application is different and so coding it is inevitable. However, the user interface components that are used with different applications remain essential the same. For example, the applications will always use a window as the base for the interface and buttons to initiate actions.

This means that if the process of building the user interface is simplified then the time and effort required in developing an application can be reduced. So it was need an environment that would allow easy designs of the user interface. Thus was born the art of visual programming.

1.11 Visual Basic Editions

- Learning Edition – It is introductory edition that lets you easily create Windows applications. It comes with all the tools you need to build main stream Windows applications.
- Professional Edition – It is for computer professionals and includes advanced features such as tools to develop ActiveX and Internet controls.
- Enterprise Edition – It is the most advanced edition and is aimed at programmers who build distributed applications in a team environment. It includes all the features of the Professional edition, plus tools such as Visual SourceSafe (a version control system) etc.

Self Learning Exercise- 3

i. Fill in the blanks

f) The ………….. Edition of Visual Basic includes ActiveX and Internet controls.

g) Visual Basic is sometimes called a ……………………… system.

h) Latest version of Visual Basic is …………

ii. State True or False

g) Visual Basic is introduced by Microsoft.

h) Visual Basic is not a visual programming language.

i) Visual programming aims at providing the user with an interface that is difficult to use.

1.12 Summary

- A file is a collection of related records.
- A file is made up of records, which are made up of fields.
- A record is recognized or identified by record KEY.
- Files can consist of records of fixed length or variable length.
- In a sequential file records are arranged in a predetermined order according to one or more keys contained in the record.
- In a direct access file the records are arranged in an order that is related to record key in a way determined by program logic.
- An organization is a group of people intentionally organized to accomplish an overall, common goal or set of goals.
- Three different levels of management are Top level management, Middle level management, Lower or operating level management.

- Visual Basic is a programming language and environment developed by Microsoft.
- Based on the BASIC language, Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces.

1.13 Glossary

RAD	Rapid Application Development (RAD) is a programming system that enables programmers to quickly build working programs.
GUI	Graphical User Interface is a program interface that takes advantage of the computer's graphics capabilities to make the program easier to use.
Internet	The Internet is a worldwide, publicly accessible network of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol (IP).
OOP	Object-oriented programming (OOP) is a programming language model organized around "objects" instead of "actions" and data instead of logic.
Object Based Language	This term is used to describe languages that support some features of Object-oriented Programming, but not all. e.g. Visual Basic supports encapsulation and polymorphism, but not inheritance.

1.14 Further Reading

1. Steven Holzner, "Visual Basic 6 Programming (Black Book)", Dreamtech Press, New Delhi.
2. Evangelas Petroutsos," Mastering Visual Basic 6", BPB Publications, New Delhi.
3. Deitel, Deitel, "Visual Basic 6 How to program", Pearson Education.
4. Noel Jerke, "The Complete Reference VB 6", TataMcgraw Hill.
5. Nambissan R., "Computerized Business Application", Galgotia, New Delhi.
6. Orilla, "An Inroduction to Business Data Processing", McGraw Hill, New Delhi

1.15 Answers to Self Learning Exercises

i. Fill in the blanks	True/False
a) relative files/ random access files	a) False
b) Sequential file, random access files	b) True
c) Fields	c) True
d) Middle Level	d) False
e) Operative management	e) False
f) Professional	f) True
g) Rapid Application Development (RAD)	g) True
h) VB.NET	h) False

1.16 UNIT END QUESTIONS

1. What are data files? Explain the concept of records and fields.
2. How is a sequential file organized? How are records in a sequential file accessed? How are these records processed?
3. Explain the random access file organization.
4. What is the concept of organization? Identify some common characteristics of any organization.
5. Discuss three different levels of management.
6. Write a note on Visual Basic.
7. List out some important features of Visual Basic.
8. Discuss the need of Visual programming languages.

Unit -2: Creating an Application in Visual Basic

Structure of the Unit

2.0 Objectives

At the end of this unit, you will be able to-

- Understand windows architecture
- Distinguish between procedural and event oriented languages
- List the various components of Visual Basic
- Understand the various programming constructs provided by Visual Basic

2.1 Introduction

In the previous unit we introduced Visual basic and discussed the need of visual programming. VB is a visual programming language.

Any VB application has two parts:

- User Interface - This is the screen displayed by the application. We interact with an application via the interface.
- Program - Computers need clear cut instructions to tell them what to do, how to do, and when to do. A set of instructions to carry out a specified task is called a program. This is what goes on in the background.

All interactions between the user and the application are via the user interface. Thus, for any application to be successful it needs to have a good user interface. A good user interface will be:

- Easy to learn
- Easy to use
- Attractive

2.2 Windows Program Architecture

Windows program does 2 things:

- Perform initial activities when the program is first loaded into memory. These activities consist of creating the program's own window and startup activities, such as setting aside some memory space.
- Process messages from windows.

The key item in the first step is creating the program's window, which is the piece of the screen that the program will control. Application programs only write inside their own window, not in other program's windows or on the background of the screen. Restricting output to the program window is one of the keys to having several programs coexist on the same screen. Program windows are always rectangular and may contain different elements such as menus, bitmaps, dialog boxes, etc., depending upon what the program does. The client area of the window is the central portion in which the program can draw graphics and text. When a program's window is visible, it will just wait until Windows sends the program a message. The waiting is accomplished by a program loop, called the 'message-loop'.

2.3 Procedural and Event Oriented Languages

Procedural languages (imperative programming) is referred to a programming paradigm based upon the concept of the procedure call. Procedures contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or by itself.

- Previous programming language paradigms have emphasized the process for solving a problem.

- Once the program was complete we would:
  - provide input
  - run the program
  - display the answer(s)
- Things always happened in the same sequence.
- The program always terminated when the process was complete.

Examples : BASIC, C, COBOL

Event-oriented language is a computer programming paradigm in which the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs. It's called event-oriented because the operating system detects events like keypresses, mouse activity, timer timeouts, and other system events, and then passes control to a program that is expecting one or more of these events to occur. Once the application processes the event, the application transfers control back to the operating system which waits for the next event to occur.

Examples : Visual Basic, Java Script, Tcl/Tk.

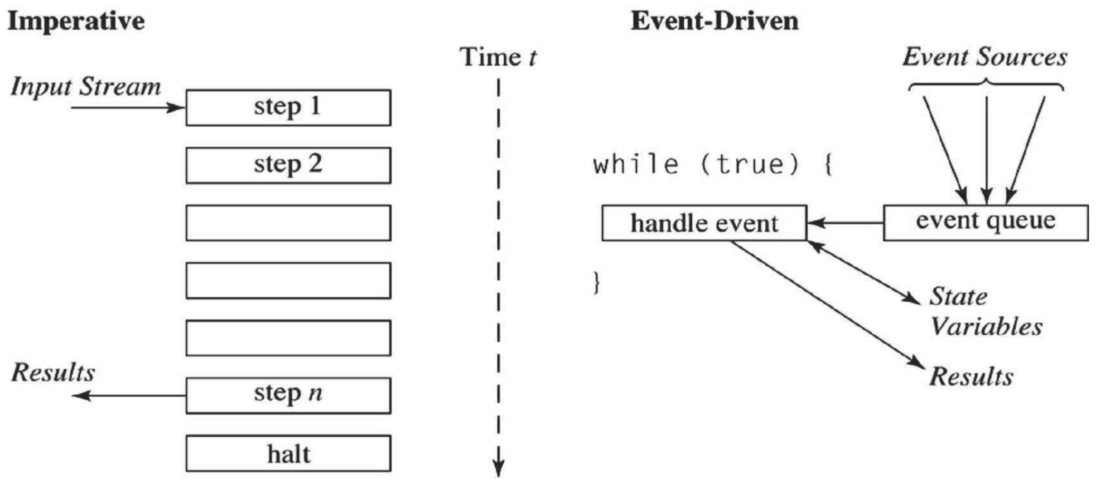


Figure 2.1 Procedural Vs Event Driven Approach

- Input to event-driven programs come from ***event sources***; sensors, input devices, objects on a web page.
- Events occur asynchronously and are placed in an ***event queue*** as they arise.
- Events are removed from the event queue and processed ("***handled***") by the program's main processing loop.
- As a result of handling an event the program may produce output or modify the value of a ***state variable***.
- There is no predefined starting or stopping point.

Self Learning Exercise -1

i. Fill in the blanks

a) Program windows are always—————.

b) Basic is a ———— language.

c) The client area of the window is the central portion in which the program can draw ———— and————.

ii. State True or False

a) Windows program accepts messages from 'message loop'.

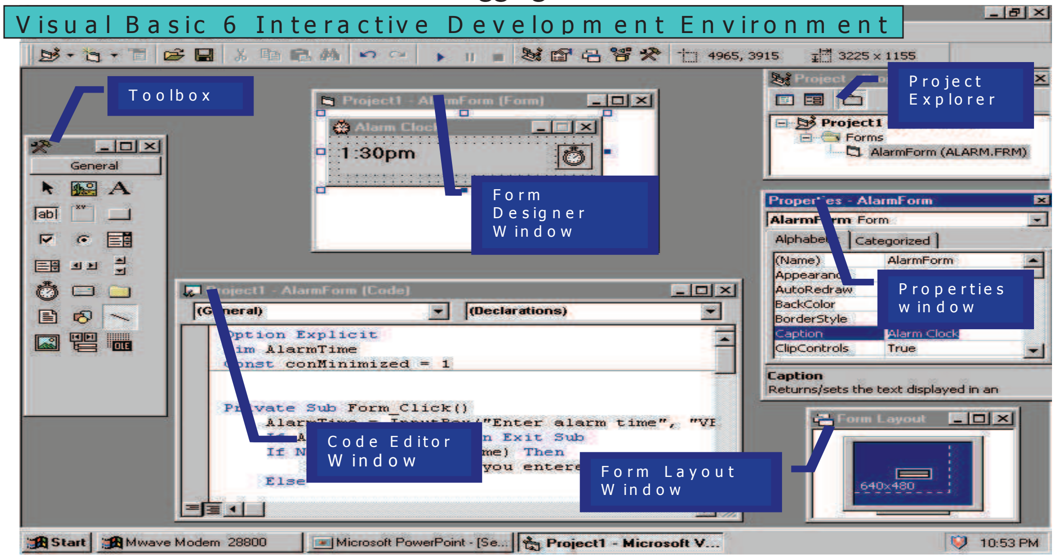
b) C is an event driven language.

2.4 Integrated Development Environment

Visual Basic is not just a language. It's an Integrated Development Environment (IDE) in which you can develop, run, test and debug your applications.

IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger. Visual Studio, Delphi, JBuilder, FrontPage and DreamWeaver are all examples of IDEs.

- **Menu Bar** – It displays the commands you used to work with VB. Each option on the menu bar has a drop-down list of items related to options chosen from it. Access key and shortcut keys are also provided.
- **Context Menu** – Contains shortcuts to frequently performed actions. Opened with right click of mouse.
- **Toolbar** – Provides quick access to commonly used commands for programming, standard functions, form design, debugging and editing.
- **Toolbox**-Provides a set of tools that you use at design time to place control on form.
- **Project Explorer Window**- It lists the forms & modules in current project (Project is the collection of files you use to build an application).
- **Properties Window**- Lists the property settings for the selected form or control. Property is the characteristic of an object.
- **Form Designer -** Displays the current form under design.
- **Form Layout-**Using this window, you can position forms as you want them to appear on the screen when they are first displayed.
- **Immediate Window** – It is a debugging aid.



(Name)	AlarmForm
Appearance	
AutoRedraw	
BackColor	
BorderStyle	
Caption	Alarm Clock
ClipControls	True

```
Option Explicit
Dim AlarmTime
Const conMinimized = 1


Private Sub Form_Click()
    AlarmTime = InputBox("Enter alarm time", "VB
    If A                    n Exit Sub
    If N                me) Then
                        you entere
    Else
```

Form Layout WiFigure 2.2 VB IDE

2.5 Types of Visual Basic Projects

An application is a collection of objects that work together to accomplish something useful. In VB the application is called a Project. A Project could be the management of a Video store, the calculation of mortgages or the Payroll for 1000 employees etc. The types of projects that can be created with Visual Basic are:

Table 2.1: Types of Visual Basic Projects

Standard EXE	The Standard EXE project allows you to create an application, which can be compiled into an .exe file. The .exe file then can be distributed using the Package & Deployment Wizard, which creates a setup program for the application.
ActiveX DLL	An ActiveX DLL's code is executed within the main program's address space. It behaves as if the class was created within the main program's code. Because the code lies inside the program's address space, calling methods is very fast. ActiveX DLL must run on the same computer as the main program.
ActiveX EXE	An ActiveX EXE's code is run in a separate process. When the main program calls an ActiveX EXE's method, the system marshals the call to translate the parameters into the ActiveX EXE's address space, calls the method, translates the results back into the main program's address space, and returns the result. This is slower than running an ActiveX DLL's method inside the main program's address space. ActiveX EXEs can run on a different computer than the main program.
ActiveX Control	An ActiveX control is a standard user interface element that you can create. These elements can be used in any container that supports ActiveX controls, such as Visual Basic forms.
ActiveX Document	An ActiveX document is a Visual Basic application that is "published" to a Web site and then downloaded and executed in the user's Web browser. An ActiveX document is not a Web page in the usual sense, as it involves no HTML or script.
DHTML Application	A DHTML application project contains one DHTML designer for each
IIS Application (Web Class)	HTML page in the project. The project can also include class modules, code modules, forms, and other executable components. When compiled, the executable components reside in a DLL file that is referenced from each HTML file in the application. This keeps the content separate from the executable components.

In Visual Basic, a project is the group of all the files that make up your program. These might include forms, modules (not attached to a form), graphics, and ActiveX controls. Common file extensions in a Visual Basic 6.0 Project are,

Table 2.2: Common file extensions in Visual Basic Projects

.vbg	Visual Basic Group project file, which contains a list of all projects in one group.
.vbp	Visual Basic Project, VB saves a project file (.vbp) in ASCII format. The project file contains entries that reflect setting for project. These include the forms & modules in a project, references, miscellaneous options that you can select to control compilation.
.frm	Form modules which contain textual description of the form and its controls including their property setting.
.bas	Standard modules, which contain public or module level declaration of the types, constants, variables, external procedures & public procedures.
.ocx	ActiveX controls are optional controls, which can be added to the toolbox and used on forms.
.cls	Class modules are similar to form modules, except that they have no visible interface. You can use class modules to create your own objects, including code for methods & properties.
.res	Resource files contain bitmaps, text strings and other data that you can change without having to re-edit your code.
.pag	The property page module file, which contains information about the property pages.
.ctl	User control modules are also similar to forms, but are used to create ActiveX controls.
.dob	ActiveX documents, that are similar to forms, but are displayable in the Internet browser such as IE.
.dsr	ActiveX designer are tools for designing classes from which objects can be created. The design interface for forms is the default design.
.exe	The executable file for a project (standard EXE)
.vbd	The ActiveX document file created to make your form web enabled.
.dll	The ActiveX DLL, or an in process component file that provides reusable code.

Self Learning Exercise -2

i. Fill in the blanks

d) Examples of IDE are————————, ———————— and——————————.

e) ——————lists the property setting for the selected form or control.

f) The executable file for a project has extension ——————.

ii. State True or False

c) VB is not an IDE.

d) Toolbar display the commands you used to work with VB.

e) The ActiveX document file has extension .vbd.

2.6 Object Models

An object model defines the structural relationships and dynamic interaction between a group of related objects, a group of objects that work together for a common purpose. Object models are in a hierarchical type of structure with one object at the root. The design of the object model defines how objects within that model can be accessed. Let us consider the following object model for a business application to be built using Visual Basic. In this model, the 'Visit' object is the root object from which all other objects are created. This example is used just to describe that an object model described the interaction between its components (objects).

In developing the business tier for the application, we are going to build a series of objects that we can classify as belonging to one of three distinct groups (fig. 2.3).

- Infrastructure
- Service
- Data

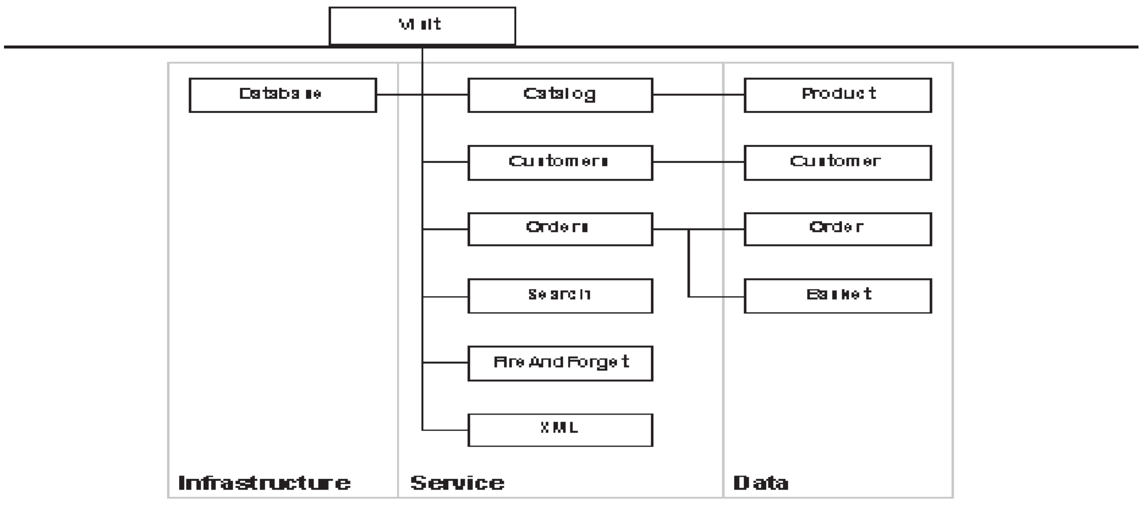


Figure 2.3 Object Model

Infrastructure Objects provide access to the resources that an application will use. In this case, we only need to actively manage one resource - the connection to the database. The application will use more resources, like memory, disk drives, and so on, but the application framework provided by Visual Basic and the Web server (either PWS or IIS) will do this for us.

One sure way of ensuring the presentation layer code is not allowed to circumvent the business rules is to never allow the code direct access to any of the infrastructure objects. In this case, we do it by creating an object that is only accessible to objects in the model. In other words, this object is private to the model as a whole and the code will not be able to directly access or call methods on it.

Service Objects provide access to application services. An application service is defined to be anything that an application can actually do. So, in this case we might have an object that can carry out operations like creating customers, deleting customers, viewing orders placed by a customer, etc. The activities outlined (for example the operation of creating a new customer) will have to conform to certain criteria - business rules.

We may decide that any object in our model can create a customer by calling the appropriate object; additionally we may decide that not only can another object call this object to create a customer but the code can as well. It is through these service objects that the presentation layer code can get to the business rules.

Data Objects define single instances of some entity in the system somewhere. This is a deliberately broad description, but in our case it nearly always refers to rows in the database. So, we may have an object to describe a single customer, or a single order, and so on. The advantage of this approach is that it lets you add a great degree of detail to the data objects.

2.7 Visual Basic Controls

A control is an object that an application uses in conjunction with a window (form) to enable user interaction. Controls provide the user with a way to type text, choose options, initiate actions, view status, and view and edit text.

2.7.1 Common Controls

- **Text Box** -The text box is the standard control that is used to receive input from the user as well as to display the output. It can handle string (text) and numeric data but not images or pictures. String in a text box can be converted to a numeric data by using the function Val(text).
- **Label** - The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. One of its most important properties is *Caption*. Using the syntax *label.caption*, it can display text and numeric data.
- **Command Button** - The command button is a very important control as it is used to execute commands. It displays an illusion that the button is pressed when the user clicks on it. The most common event associated with the command button is the Click event, and the syntax for the procedure is

  ```
  Private Sub Command1_Click ()
          Statements
  End Sub
  ```
- **Picture Box** -The Picture Box is one of the controls that is used to handle graphics. You can load a picture at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. You can also load the picture at runtime using the *LoadPicture* method. For example, the statement will load the picture grape.gif into the picture box.

  Picture1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")

  The image in the picture box is not resizable.
- **Image Box**- The Image Box is another control that handles images and pictures. It functions almost identically to the picture box. However, there is one major difference, the image in an Image Box is stretchable, which means it can be resized. This feature is not available in the Picture Box. Similar to the Picture Box, it can also use the *LoadPicture* method to load the picture. For example, the statement loads the picture grape.gif into the image box.

  Image1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")
- **List Box** - The function of the List Box is to present a list of items where the user can

click and select the items from the list. In order to add items to the list, we can use the *AddItem* method.

```
List1.AddItem "Lesson1"
List1.AddItem "Lesson2"
```

The items in the list box can be identified by the *ListIndex* property, the value of the *ListIndex* for the first item is 0, the second item has a *ListIndex* 1, and the sec-ond item has a *ListIndex* 2 and so on

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

  ```
  Combo1.AddItem "Item1"
  ```

- **Check Box** -The Check Box control lets the user select or unselect an option.

  You can include the statements Check1.Value=1 to mark the Check Box and Check1.Value=0 to unmark the Check Box, and use them to initiate certain actions. For example, the program will change the background color of the form to red when the check box is unchecked and it will change to blue when the check box is checked.

  ```
  Private Sub Check1_Click ()
      If Check1.Value = 0 Then
          Form1.BackColor = vbRed
      ElseIf Check1.Value = 1 Then
          Form1.BackColor = vbBlue
      End If
  End Sub
  ```

- **Option Box** - The Option Box control also lets the user selects one of the choices. However, two or more Option Boxes must work together because as one of the Option Boxes is selected, the other Option Boxes will be unselected.

  In the following example, the shape control is placed in the form together with six Option Boxes. When the user clicks on different option boxes, different shapes will appear. The values of the shape control are 0, 1, and 2,3,4,5 which will make it ap-pear as a rectangle, a square, an oval shape, a rounded rectangle and a rounded square respectively.

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

```
End Sub
Private Sub Option4_Click()
        Shape1.Shape = 3
End Sub
Private Sub Option5_Click()
        Shape1.Shape = 4
End Sub
Private Sub Option6_Click()
        Shape1.Shape = 5
End Sub
```

- **Drive List Box** -The Drive ListBox is used to display a list of drives available in your computer. When you place this control into the form and run the program, you will be able to select different drives from your computer.
- **Directory List Box** -The Directory List Box is used to display the list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer.
- **File List Box** -The File List Box is used to display the list of files in a selected directory or folder. When you place this control into the form and run the program, you will be able to a list of files in a selected directory.

2.7.2 Other Controls

- Frame

  In Visual Basic 6.0, the Frame control is used as a container for grouping controls.
- Timer

  Creates a timer that can execute code at regular intervals. The Timer control is an actual control that is sited on a form at design time; however it is not visible at run time.
- Shape

  The Shape control provides an easy way to draw rectangles, circles, and other shapes on a form at design time.
- Line

  The Line control provides an easy way to draw lines on a form at design time.
- Data

  The Visual Basic 6.0 Data control is used as a mechanism for binding controls to a database using DAO. The Data control also provides an interface for navigating data, with buttons for moving back and forth through rows in a database table.
- OLE

  The OLE Container control is used to add insertable OLE objects to forms.

Table 2.3: Properties and Events Common to Controls

Common Events	**Common Properties**
● Mouse Events ➢ Click ➢ DblClick ➢ MouseDown ➢ MouseUp ➢ MouseMove	● Name- Name of control ● Appearance- 0 for flat look & 1 for 3-D look. ● BackColor ● ForeColor ● Font
● Keyboard Events ➢ KeyDown ➢ KeyUp ➢ KeyPress ➢ Change	● Caption ● Text ● Width, Height ● Left, Top – set the coordinates of the control's upper-left corner
● Focus ➢ GotFocus ➢ LostFocus	● Enabled- True (control can get focus), false(disable the control) ● Visible

Self Learning Exercise -3

i. Fill in the blanks

g) Service objects provide access to ——————————..

h) ————————are used to accept information from the user.

i) The —————————— is used to display the list of directories or folders in a selected drive.

ii. State True or False

f) The window is the base for user interface of an application.

g) Label is used to receive input from the user as well as to display the output.

h) An event is any user action directed at the application.

2.8 Variables and Constants

Variable is a location in the computer's memory where data is stored, it is a storage location with a type. The type is one of the intrinsic Visual Basic types. Variables can have values assigned to them and these values can be changed programmatically. Variables are a powerful tool but, there are times when we want to manipulate a defined value, one whose value you want to ensure remains constant.

2.8.1 Managing Variables

Variables are used to store values. To name a variable in Visual Basic, you have to follow a set of rules. The following are the rules when naming a variable 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

Examples of valid and invalid variable names are displayed below:

Valid Name **Invalid Name**

My_Car	My.Car
this year	1NewBoy
Long_Name_Can_beUSED	He&HisFather (& is not acceptable)

In Visual Basic, one needs to declare the variables before using them by assigning names and data types. They are normally declared in the general section of the code's window using the **Dim** statement. The format is as follows:

Dim variableName as DataType

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

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 following are some examples:

FirstNumber=100

SecondNumber=firstNumber-99

UserName="John Lyan"

UserPass.Text = password

Label1.Visible = True

2.8.2 Managing Constants

Constant is a variable that has fixed value**.** To declare a constant you use the following syntax:

Scope Const ConstantName As DataType = Value

Scope represents the range of the variable. ConstantName is a string representing the name of the variable. DataType represents what sort of data is stored in the variable. Value is the value of the constant. For example,

Public Const MissingDiskMsg As String = "Disk not ready. Please insert a disk"

if declared in the General Declarations section of a form, MissingDiskMsg can be accessed in all procedures in that form. So

Msgbox MissingDiskMsg, vbOKOnly

Displays a message box saying 'Disk not ready. Please insert a disk.'.

2.9 Visual Basic Data Types

Visual Basic offers the usual set of intrinsic (built-in) data types one expects in a modern language. Each type has a specific and unchanging size. The following table illustrates the data types available in Visual Basic with their storage size and the range of values they can accommodate.

Table 2.4 Visual Basic Data Types

Data type	Storage size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any Object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a Double
Variant (with characters)	22 bytes + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type.

2.10 Operators in Visual Basic

In order to compute inputs from users and to generate results, we need to use various mathematical operators. In Visual Basic, the symbols for the operators are somewhat different from normal mathematical operators, as shown in Table 2.5.

Table 2.5: Arithmetic Operators

Operator	Mathematical function	Example
^	Exponential	2^4=16
*	Multiplication	4*3=12
/	Division	12/4=3
Mod	Modulus(returns the remainder from an integer division)	15 Mod 4=3
\	Integer Division(discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

```
Dim firstName As String
Dim secondName As String
Dim yourName As String
 Private Sub Command1_Click()
firstName = Text1.Text
secondName = Text2.Text
yourName = secondName + “  “ + firstName
      Label1.Caption = yourName
End Sub
```

Self Learning Exercise -4

i. Fill in the blanks

j) ——————data type can store any object reference.

k) We can declare any variable using ——————.

l) To store variable length data you can use...........data type.

ii. State True or False

i) Variable name must be greater than 255 characters.

j) Scope represents the range of the variable.

2.11 Summary

- Procedural languages are based upon the concept of the *procedure call*.
- Event-oriented language is a computer programming paradigm in which the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs.
- Visual Basic is not just a language. It’s an Integrated Development Environment in which you can develop, run, test and debug your applications.
- Object models are in a hierarchical type of structure with one object at the root.

- A Visual Basic programming environment provides all features that are required to develop a graphical user interface as ready to use components.
- Visual programming commonly uses windows, buttons, text boxes, list boxes as components.
- The characteristics of the components are referred to as its properties.
- A method is a code that is built into component and can be executed as required.
- An event is any user action directed at the application.
- Variables store values required for performing calculations.

2.12 Glossary

HAL	*Hardware Abstraction Layer* (HAL) is a layer of programming that allows a computer operating system to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.
SRM	*Security Reference Monitor* Checks for proper authorization before granting access to objects.
DLL	*Dynamic Link Library*, a library of executable functions or data that can be used by a Windows application
Windows NT	Windows NT (New Technology) is a family of operating systems produced by Microsoft
ActiveX	A loosely defined set of technologies developed by Microsoft for sharing information among different applications

2.13 Further Readings

1. Steven Holzner, "Visual Basic 6 Programming (Black Book)", Dreamtech Press, New Delhi.
2. Evangelas Petroutsos, "Mastering Visual Basic 6", BPB Publications
3. Deitel, Deitel, "Visual Basic 6 How to program", Pearson Education
4. Noel Jerke, "The complete reference VB 6", TataMcgraw Hill

2.14 Answers to Self Learning Exercises

i. Fill in the blanks	ii. True/False
a)rectangular	a)True
b)procedural	b)False
c)graphics, text	c) False
d)Visual Studio, FrontPage, DreamWeaver	d) False
e)Properties Window	e)True
f).exe	f)True
g)application services	g)False
h)Text Boxes	h)True
i)Directory List Box	i)False
j)object	j)True
k)dim	
l)Variant	

2.15 Unit End Questions

1. Discuss the main layers of Window Architecture.
2. Distinguish procedural and event driven languages. Why event driven languages are so popular? Give examples of both.
3. What do you understand by IDE? What are the advantages of IDE? Discuss main elements of IDE.
4. Create Object Model for any restaurant.
5. Design user interface in VB that accepts student name(textbox), class(Combo box), marks in 3 subjects(text box) and then display report card in text box in following format

   Name: Ram Class : BCA I year Grade : A

Unit - 3: Writing Code in Visual Basic

Structure of the Unit

3.0 Objectives

At the end of the unit, you will be able to

- Invoke Visual Basic
- Create and run the application
- Understand basic decision control structure (if-else, select)
- Understand loop control structure (Do-while, For, While)
- Understand menus, sub-procedures and sub-functions
- List the properties, methods and events of most commonly used controls in Visual Basic
- Decide what controls are to be used and when

3.1 Introduction

In the previous chapter, we were introduced to the Visual Basic. This chapter focuses on the some basic building blocks of VB such as loops, built in functions, menus.

Then we discuss some basic controls used in user interface of VB. Each interface element was discusses in detail in terms of:

Where is it used?

- Important Properties
- Important Methods
- Important Events

Finally the chapter discusses the types of data validation.

3.2 Invoking Visual Basic

To invoke Visual Basic, follow these steps:

- Click on the 'Start' button.
- Select the option 'Programs'.
- Select the option 'Microsoft Visual Studio 6.0' from the options display.
- Click on the option 'Microsoft Visual Basic 6.0' from those displayed.
- Click on the 'Open' button. (The Standard EXE option is selected by default.)

A new project always provides a form. A form is a window in Visual Basic application.

There are two main steps in creating an application in Visual Basic:

1. ***Design the interface***- The first step is to create the forms. It has two stages: placing controls in the form and setting the control properties. When you drag a control from Toolbox, corresponding properties will be displayed in Property Window.
2. ***Write code***- After designing the interface the next step is to associate the code with the controls. When you double click on the Command button1, the Code Window displays the following lines:

```
Private Sub Command1_Click()
    'write your code here
End Sub
```

3.2.1 Saving the Project and Running

Saving a project can be complicated in Visual Basic because it involves saving multiple files. To save a new Visual Basic project for the first time,

- Select the sub option 'Save Project' in 'File' option.

You will be prompted separately for a form name (i.e. the name of the *.frm* file) and then a project name (the *.vbp* file). Usually same name is given to both the files.

- Save the form first and then save the project.

There should be one Visual Basic Project (.VBP) file and separate Form (.FRM) and Module (.BAS) files for each form and module used in the current project.

To execute a Visual Basic Project, click on the start button on the toolbar or select start from the run menu.

- Select the menu option 'Run'.
- Click on the sub option 'Start'.

3.3 Decision Control Structure

Visual Basic includes a number of features that allow us to select among alternative pathways or to repeat the execution of a particular block of statement. For example, we can select to execute one of several different blocks or two blocks of statements, depending on value of an expression. This process is known as selection/ branching and the statements that are used in selection are, If.....Then.....Else and Select…..Case. Many programs require that a set of statements be executed repeatedly, until some particular condition is satisfied. This process is known as looping. The statements used for looping are Do…While/Until, For…Next and While… Wend

3.3.1 Using If.....Then.....Else Statements with Operators

To effectively control the Visual Basic program flow, we use If...Then...Else statement together with the conditional operators and logical operators. If…Then…Else execute a expression if a condition is true and execute a different expression if it is false.

The general format for the if...then...else statement is

```
If condition(s) Then
   VB expressions
Else
   VB expressions
End If
```

Any If…Then...Else statement must end with End If. Sometimes it is not necessary to use Else.

Example

The following example accepts a number from the user and multiplies it by 2.

```
Private Sub Form_Click()
        Dim number As Integer, ans As Integer
        number = InputBox("Enter a number")
        If number >0 then
          ans = number * 2
```

```
            Print ans
          End If
End Sub
```

3.3.2 Select…..Case

If you have a lot of conditional statements, using If…Then…Else could be very messy. For multiple conditional statements, it is better to use Select…Case. Select…Case allows us to do various comparisons against one variable that we use throughout the whole statement.

The format is:

```
Select Case expression
 Case value1
       Block of one or more VB statements
   Case value2
       Block of one or more VB Statements
   Case value3
       Block of one or more VB statements
   Case value4
       .
       .
       .
   Case Else
       Block of one or more VB Statements
End Select
```

The data type specified in expression must match that of Case values.

Examples

Example 1:

```
grade=txtgrade.Text
Select Case grade
  Case  "A"
      result.Caption="High Distinction"
      .
      .
End Select
```

Example 2:

```
mark = mrk.Text
Select Case mark
 Case Is >= 85
      comment.Caption = "Excellence"
```

Example 3:

```
 mark = mrk.Text
Select Case mark
Case 0 to 49
       comment.Caption = "Need to work harder"
```

3.4 Loop Control Structures

Visual Basic allows a procedure to be repeated as many times as long as the processor could support. This is generally called looping .

3.4.1 Do Loop

The Do loop can be used to execute a fixed block of statement indefinite number of times. The Do loop keeps executing it's statement while or until the condition is true. Two keywords while and until can be used with the do loop.

The format are

```
a)  Do While condition
          Block of one or more VB statements
     Loop
b)  Do
          Block of one or more VB statements
     Loop While condition
c)  Do Until condition
          Block of one or more VB statements
     Loop
d)  Do
          Block of one or more VB statements
     Loop Until condition
```

3.4.2 For....Next Loop

The For loop is the most popular loop. For loops enables us to execute a series of expressions multiple numbers of times.

The format is:

```
 For counter=startNumber to endNumber (Step increment)
        One or more VB statements
 Next
```

Example:

```
(a)     For counter=1 to 10
              display.Text=counter
        Next
(b)     For counter=1 to 1000 step 10
              counter=counter+1
        Next
```

3.4.3 The While…Wend loop

While loop is same as do loop. It can be used to execute a fixed block of statement indefinite number of times. The While loop executes a series of statements as long as a given condition is True.

```
While condition
        statement
Wend
```

3.5 Sub Procedures

Procedures are a series of statement that are executed when called. Sub procedures are procedures which do not return a value. Each time when the Sub procedure is called, the statements within it are executed until the matching End Sub is encountered. It can be defined with the "Sub" statement:

```
Sub sub_name(argument_list)
        statement_block
End Sub
```

Example:

```
Sub Showdate()
        MsgBox Date()
End Sub
```

3.6 Visual Basic Functions

Function is a Procedure which returns a value. They are similar to normal procedures but the main purpose of the functions is to accept certain inputs and pass them on to the main program to finish the execution. Functions are declared with the keyword Function. In VB there are two types of functions, inbuilt and user defined. Some of inbuilt functions are, InputBox(), MsgBox(), Left(), Right() etc.

3.6.1 Inbuilt functions

A function is referred to as "built-in" if it shipped with your language. To make your job a little easier, Microsoft Visual Basic comes equipped with many functions that you can use right away in your program. Some of inbuilt functions are, InputBox(), MsgBox(), Left$(), Right$() etc. Some of these functions will be discussed in ore detail in the next chapter.

3.6.2 User Defined Functions

The general format of a function is as follows:

Public Function functionName (Arg As dataType,..........) As dataType

or

Private Function functionName (Arg As dataType,..........) As dataType

*Public indicates that the function is applicable to the whole program and

*Private indicates that the function is only applicable to a certain module or procedure.

```
Function function_name(argument_list)
        statement_block
        function_name = return_value
End Function
```

Example:

```
Function NextDay() As Date
        NextDay = Date()+1
End Function
```

Passing argument by Value

By value is a way in which you pass values and protect the calling procedure's passed data so that the called procedure cannot change the original data. To pass argument as call by value, use ByVal.

Example:

```
Function Add(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
  Add = n1 + n2
  n1 = 0
  n2 = 0
End Function
```

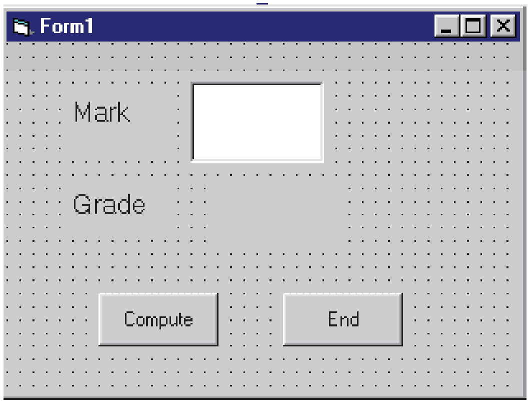
Passing argument by Reference

By reference is a way in which you pass values and allow the called procedure to change those values, also called by address. Default parameter passing technique in VB.

Example:

```
Function Add(n1 As Integer, n2 As Integer) As Integer
  Add = n1 + n2
  n1 = 0
  n2 = 0
End Function
```

The following program will automatically compute examination grades based on the marks that a student obtained.



```
Public Function grade(mark As Variant) As String
```

```
Select Case mark
    Case Is >= 80
        grade = "A"
    Case Is >= 70
        grade = "B"
    Case Is >= 60
        grade = "C"
    Case Is >= 50
        grade = "D"
    Case Is >= 40
        grade = "E"
    Case Else
        grade = "F"
End Select
End Function
Private Sub compute_Click()
    grading.Caption = grade(mark)
End Sub
Private Sub End_Click()
    End
End Sub
```

Self Learning Exercise-1

i. Fill in the blanks

a) The ———— function will display a message box where the user can enter a value or a message in the form of text.

b) The objective of———— is to produce a pop-up message box.

c) The style value of VbOk is—.

d) We can pass arguments in function by——and by ————.

ii. State True or False

a) Message boxes display output, and input boxes get input.

b) If you have a lot of conditional statements, If-Then-Else is better.

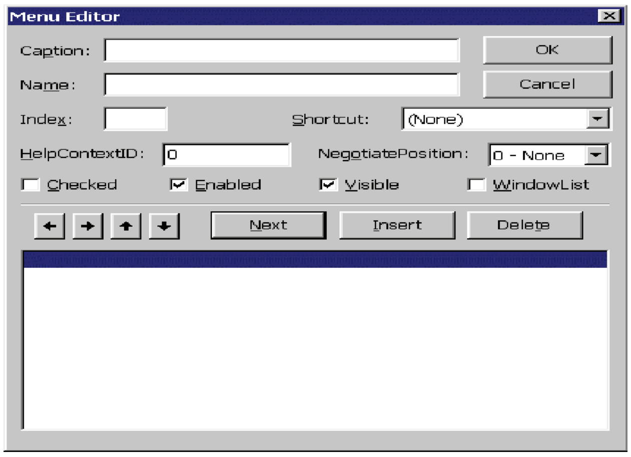
c) Default parameter passing technique is by value.

3.7 Menus

Menus are often necessary in applications and it is essential that developers have an ability to manipulate them in design-time. Creating menus is fairly simple in VB. Once a menu item is created, it acts about the same as a Command Button. You simply program its click event.

3.7.1 Creating menus

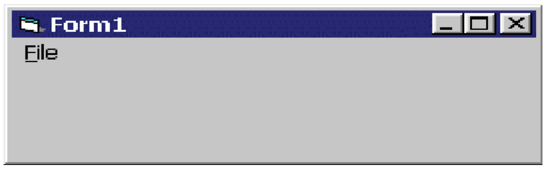
To create menus in VB you use the Menu Editor. You can find this under Tools | Menu Editor. The VB Menu editor:



Menu Editor

To create a top menu (i.e. File), use the following steps:

1) Select Tools | Menu Editor
2) Enter the text to be displayed for the menu in the Caption text box (i.e. File). As always you can use the & to set the Shortcut key: So &File would display the menu File and could be accessed by Alt+F
3) Enter a name for the menu. This will be used when accessing the menu's properties in your code.
4) Make sure that there are no dots before File in the list box (i.e. ...File). If there are, press the < button on the dialog box until there are none. These will be explained later.
5) Click OK. If you have used the examples, you will now see something like this:

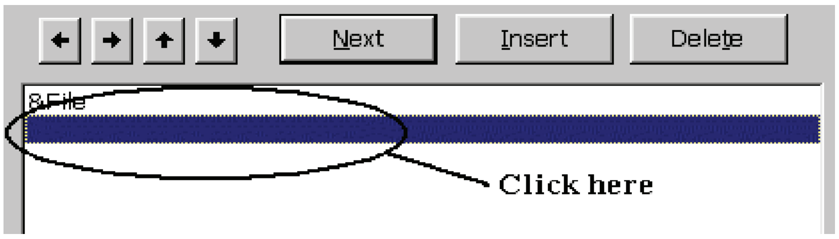


Main Menu

3.7.2 Creating menu items

To create a menu item (i.e. Open, on the File Menu), use the following steps:

1) Select Tools | Menu Editor
2) Click the item on the list after the menu you want this item to go on. (i.e. after File). If there is an item already there, click Insert:

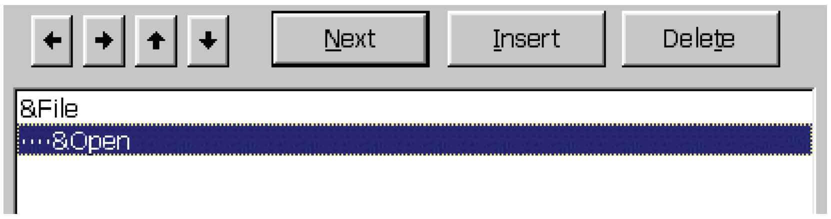


Options in Menu Editor

3) Enter the text to be displayed for the menu item in the Caption text box (i.e. Open). As always you can use the & to set the Shortcut key: So &Open would display the menu item Open and could be accessed by Alt+O when the File menu is open

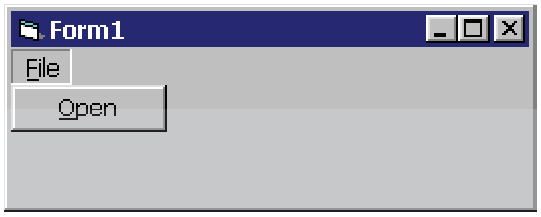
4) Enter a name for the menu item. This will be used when accessing the menu's properties in your code.

5) Press the > button once. You will see four dots (....) appear before the text Open. This shows that Open is a item on the menu File:



Creation of sub menu

6) Click OK. If you have used the examples, you will now see something like this when you click on File:



File sub menu

6) To write some code that runs when a menu item is clicked, simply click the menu item during design time. A code window will open with a new procedure. Enter any code you want to run when the menu item is clicked here.

3.7.3 Creating Submenus

In the Menu editor, you will notice that when you press the < and > buttons, dots will appear before the selected item. These show what item belongs to what. For example:

&File	<< Top Menu
....&Open	<< Menu item of File
....&Save	<< Menu item of File
....&Insert...	<< Menu item of File
........&Text	<< Sub Menu of Insert
........&Image	<< Sub Menu of Insert
....&Close	<< Menu item of File
&Help	<< Top Menu
....&About	<< Menu item of Help

So, to create a sub menu, simply follow these steps:

1) Select Tools | Menu Editor

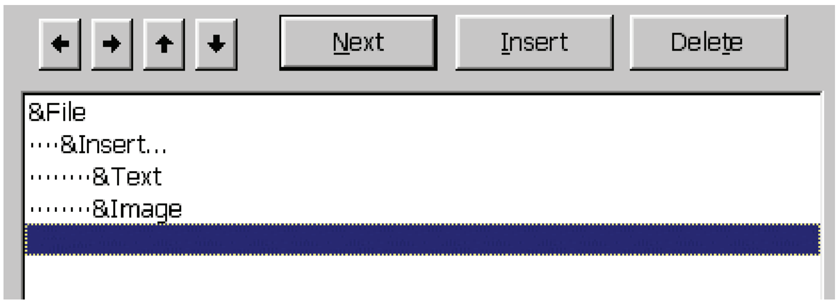
2) Create a menu item which you want to contain the sub menu (which will have the > after it) i.e. Insert on the File menu

3) Insert an item after the menu item you have just created. (i.e. after Insert...). Click Insert:

4) Enter the text to be displayed for the sub menu item in the Caption text box (i.e. Text). As always you can use the & to set the Shortcut key: So &Text would display the menu item Text.

5) Enter a name for the menu item. This will be used when accessing the menu's properties in your code.

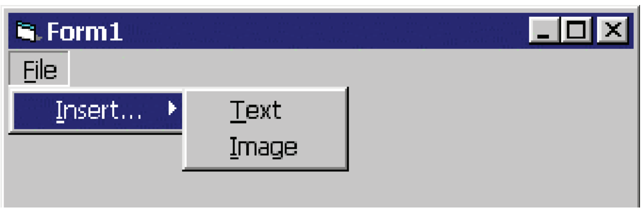
6) Press the > button until the item is indented once more than the menu you want to contain a sub menu. You will see four dots (....) appear before the text Open. This shows that Open is a sub menu item on the menu item Insert:



&File
····&Insert...
········&Text
········&Image

How to create submenu in submenu

6) Click OK. If you have used the examples, you will now see something like this when you click on File:



Menu control list box

Self Learning Exercise-2

i. Fill in the blanks

e) To create menus in VB you use the……..

f) &File would display the menu File and could be accessed by ……….

g) We can use the & to set the ....

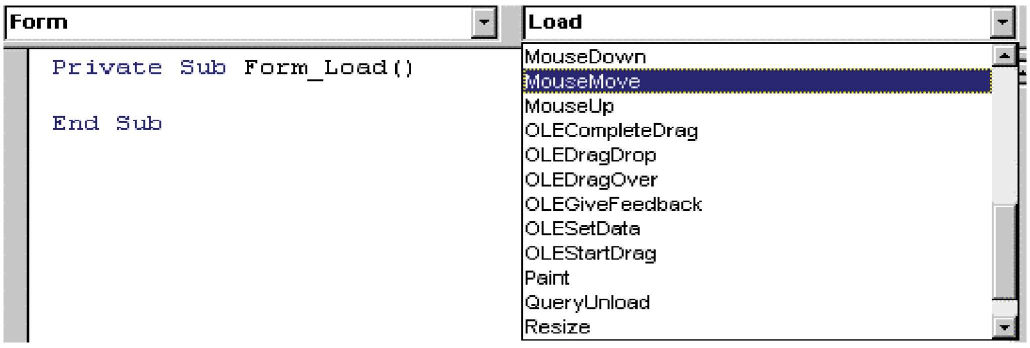
ii. State True or False

d) Caption is used to accessing the menu's properties in your code.

e) In menu editor —> button is used to insert sub menu.

3.8 What is an event?

The 'look' of a Visual Basic application is determined by what controls are used, but the 'feel' is determined by the events. An event is something which can happen to a control. For example, a user can click on a button, change a text box, or resize a form. As explained in Creating a Visual Basic Application, writing a program is made up of three events: 1) select suitable controls, 2) set the properties, and 3) write the code. It is at the code writing stage when it becomes important to choose appropriate events for each control. To do this double click on the control the event will be used for, or click on the icon in the project window (usually top right of screen). A code window should now be displayed similar to the one shown below.



```
Private Sub Form_Load()

End Sub
```

Events of Form Control

The left hand dropdown box provides a list of all controls used by the current form, the form itself, and a special section called General Declarations. The corresponding dropdown box on the right displays a list of all events applicable to the current control (as specified by the left hand dropdown box). Events displayed in bold signify that code has already been written for them, unbold events are unused.

Common Events are:

DoubleClick: This event is triggered whenever a user double clicks on that control.

MouseMove: When a user passes the mouse over a control this event is triggered.

GetFocus: This event is triggered when you first give focus to a control either by clicking on it or by tabbing to it.

LostFocus: This event is triggered when you have had focus on a control and then attempt to move focus from that control.

Load: This event is only in the form control and it is initiated when the form is first loaded.

3.9 Common controls

- **The Form**

  The window is called a form in VB. It is the base on which the user interface is built.

Table 3.1 Form Methods

Method	Description	How to use
Load	The Load statement initializes and loads the form into memory without displaying it on the screen.	Load Form1
Unload	Unload removes a form from memory.	Unload Form1 or Unload Me
Show	The Show method loads a form into memory if not loaded already and displays it on the screen.	Form1.Show
Hide	The Hide method removes a form from the screen.	Form1.Hide / Me.Hide

Table 3.2 Form Properties and Events

Properties		Events
Caption :- The text that appears on the title bar of the form. Name:- Text used in the code to identify a form		Initialize/ Terminate Load/ Unload Activate/ Deactivate QueryUnload

- **The List Box**

  List boxes display long lists of options from which users can choose.

Table 3.3 List Box Properties

List Index	Is a number used to access individual elements in the list box (starts with 0).
Sorted	Used to display the items in a sorted order.
MultiSelect	Used to specify if the user can select multiple items in the list.
ListCount	Used to return the number of items in a list box
Selected	Sets the selection status of an item in a ListBox control.

Table 3.4 List Box Methods

AddItem	This method adds the specified item to the list box.
RemoveItem	Used to delete an item from the list.
SetFocus	Used to make the list box the current active element

Table 3.5 List Box Events

Click	Occurs each time the user clicks on the list box.
Scroll	Occurs when a user scrolls through the list in the list box.

- **The Combo Box or Dropdown List Box**

  A combo box control combines the features of a text box and a list box.

Table 3.6 Combo Box properties

Style	The value assigned to this property decides the "look" of the combo box. Three styles: Simple Combo Box Drop down Combo Box Drop down List Combo Box
Locked	Used to specify whether the user can enter a value in the text box section of the control.

- **Option Button (Radio Button)**

  The radio buttons are used when the user can select one and only one of the multiple options.

Table 3.7 Properties of Option Button

Properties		Method	Events
Caption- The text that appears next to the option button. **Value**- This property specifies whether the option button is selected. **Enabled**- Sets a value that determines whether a form or control can respond to user-generated events.		**Move** – Move a control on the form.	**Click**- Occurs when the user clicks on the option button.

- **The Frame Control**

Table 3.8 Properties of Frame control

Properties		Method	Events
Caption – The text that appears on the frame.		Move - Moves the frame control along with the controls placed within it. Drag – Begin, ends or cancels a drag operation of the control	Click – Occurs when the user clicks on the frame control.

- **The Check Box**

Table 3.9 Properties of Check Box

Properties	Method	Events
Caption– The text that appears on the check box. Value- Where the check box is selected	Move	Click – Occurs when the user clicks on check box.

- **The Scroll Bars**

Scroll bars provide easy navigation through a log list of items or a large amount of information. The values can range from –32,768 to 32,767.

Table 3.10 Properties of Scroll Bars

Properties	Method	Events
Min- Defines the smallest value. Max- Defines the largest value. Value- Current value of scroll bar.	Move	Scroll- Occurs when the scroll bar control is repositioned Change- Occurs when the user scrolls through the scroll bar.

Self Learning Exercise-3

i. Fill in the blanks

h) ————— display long lists of options from which users can choose.

i) ————— event is triggered when you first give focus to a control.

j) ————— provide easy navigation through a long list of items or a large amount of information.

k) The window is called a ———— in VB.

ii. State True or False

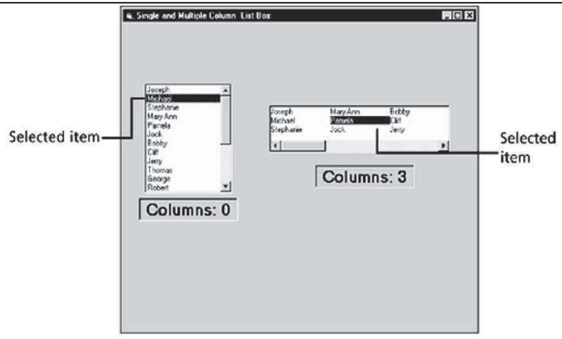
f) The text that appears on the title bar of the form is called the caption.

g) The window is called a frame in VB.

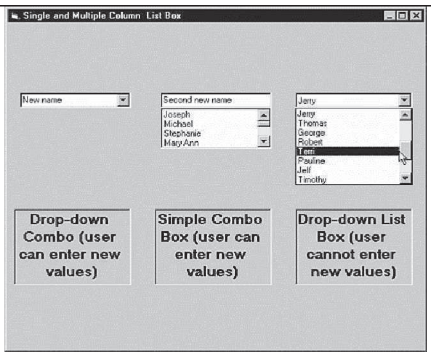
h) Load and Unload are Form methods.

i) Check boxes are used when the user can select one and only one of the multiple options.

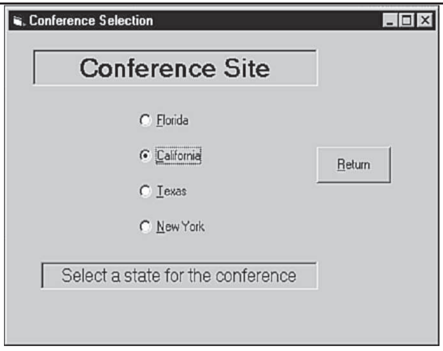
Figures of common controls



List Box

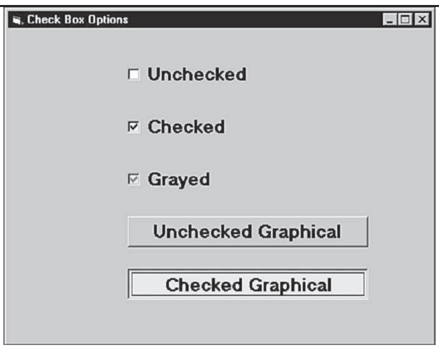

Single and Multiple Column List Box
New name
Second new name
Joseph
Michael
Stephanie
MaryAnn
Jerry
Jerry
Thomas
George
Robert
Terri
Pauline
Jeff
Timothy
Drop-down Combo (user can enter new values)
Simple Combo Box (user can enter new values)
Drop-down List Box (user cannot enter new values)

Combo Box

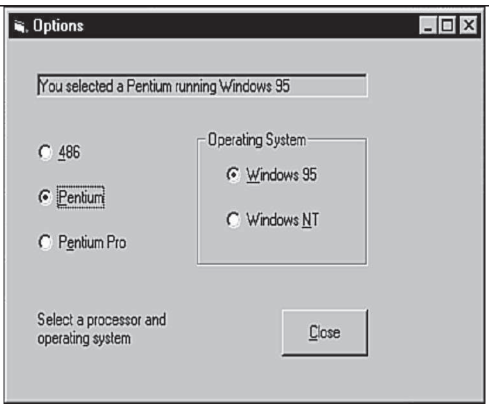


Select a state for the conference

Option Buttons



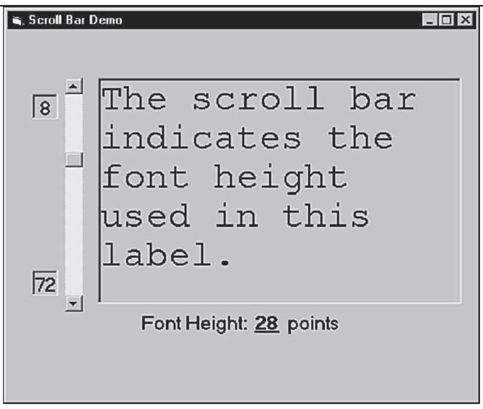
Check boxes



You selected a Pentium running Windows 95

Select a processor and operating system

Frame Control



The scroll bar indicates the font height used in this label.

Scroll Bar

3.10 DATA VALIDATION

In Visual Basic, there are two data validation techniques:

Form-Level Validation: Data is verified after the user enters all the fields on the form.

Field-Level Validation : Data in the fields are verified one at a time, after the user fills in the fields.

3.10.1 Field-Level Validation

Field-level validation ensures that the value entered in the field as a whole is in accordance with the application's requirement. If it isn't, you can alert the user to the problem. Appropriate times to perform field-level validations are

- When the user attempts to leave the field.
- When the content of the field changes for any reason. This isn't always a feasible strategy. For example, if you're validating a date to be formatted as "mm/dd/yy", it won't be in that format until all the keystrokes are entered.

When a user enters and leaves a field, events occur in the following order:

1. Enter
2. GotFocus
3. Leave
4. Validating
5. Validated
6. LostFocus

3.10.2 Form-Level Validation

Form –Level validation is the process of validating data in all the fields on a form. We usually write code to validate data in the Click event of the CommandButton, after the user has entered all the fields on the form.

When we implement form-level validation, we verify the data on a form in a single procedure.

The three main events used in form-level validation are: KeyPress, KeyDown, KeyUp

Some validation examples are:

1. LostFocus event

When a form or a control loses focus, the LostFocus event is fired. Focus is lost when a user tabs to another fields or clicks on another filed. We are checking the data just as the user is leaving the field.

To verify that the employee code consists of 4 characters

```
Private Sub txtcode_LostFocus()
   If Len ( txtCode.Text) <> 4 Then
     Beep
    MsgBox "code should consist of 4 characters"
     txtCode.SetFocus
   End If
 End Sub
```

2. SetFocus Method

In form-level and field-level validations focus is always set back to the field with invalid data.

For example, if the user leaves the name of the employee blank we use the SetFocus method to place the cursor on the TextBox.

```
If txtName.Text = "" Then
      Beep
     MsgBox "Name should not be empty"
      txtName.SetFocus
End If
```

Self Learning Exercise-4

i. Fill in the blanks

l) ———————— validation is the process of validating data in all the fields on a form.

m) When a form or a control loses focus, the ———— event is fired.

n) The three main events used in form-level validation are: ————, ———— and —————

ii. State True or False

j) In form-level and field-level validations focus is always set back to the field with invalid data.

k) In field-level validation data in the fields are verified every time, after the user fills in the fields.

3.11 Summary

- Visual Basic offers several forms of the If and the Select…Case statements to make comparisons.
- The comparison operators, especially when combined with the logical operators, produce advanced compound conditions.
- Message boxes display output, and input boxes get input.
- The message and input boxes offer ways for your programs to request information that regular controls can't handle.
- Use controls to display and get data values that are always needed.
- Use message and input boxes to display messages and get answers that the program needs in special cases, such as for error conditions and exception handling.
- Visual Basic supports several forms of looping statements. The Do and For loops provide you with the power to write any kind of looping section your program needs.
- The first kind of list, the List Box control, lets users select from a list of items that an application displays.
- The Combo Box control works like a List Box control but lets the user enter new values into the list.
- The option buttons and check boxes work almost exactly alike except that the option buttons are mutually exclusive and provide users with a single option from a selection.
- The scrollbars let users select values based on a range, using either a horizontal or verti-

cal scrollbar.

3.12 Glossary

EXE	**EXE** is the common filename extension for denoting an executable file (a program) in the OpenVMS, MS-DOS, Microsoft Windows, and OS/2 operating systems.
VBP	Visual Basic Project file. The *.VBP file provides a lot of information about a Visual Basic project, including the type of executable it creates, the references and components it contains, and the information included under the Project Properties popup Make tab.
Menu Editor	Menu Editor is an editor to create menus in Visual Basic.

3.13 FURTHER REFERENCES

1. Steven Holzner, “Visual Basic 6 Programming (Black Book)”, Dreamtech Press, New Delhi.
2. Evangelas Petroutsos,”Mastering Visual Basic 6", BPB Publication
3. Deitel, Deitel, “Visual Basic 6 How to program”, Pearson Education
4. Noel Jerke, “The complete reference VB 6”, TataMcgraw Hill

3.14 Answers to Self Learning Exercises

i. Fill in the blanks	ii. True/False
a) InputBox()	a)True
b) MsgBOx	b) False
c) 1	c) False
d) value, reference	d) False
e) Menu Editor.	e) True
f) ALT+F	f) True
g) Shortcut key	g) False
h) List Box	h)True
i) GetFocus	i)False
j) Scrollbar	j) True
k) form	k) False
l) Form-Level	
m) Lost Focus	
n) KeyPress, KeyDown, KeyUp	

3.15 Unit End Questions

1. What is the difference between ‘Caption’ property and ‘Name’ property?
2. What is the difference between List and ListIndex property?
3. Discuss VB in built functions?
4. List out common properties, methods and events of List Box and Combo Box.
5. Why VB is known as event driven programming language?
6. What is the importance of data validation? Discuss different types of data validation in VB?

Unit - 4: IDE, Forms and Controls

Structure of the Unit

4.0 Objectives

IDE means software in which there is an environment provided to user in which there are many tools available for coding, testing, debugging, deployment and maintenance.

The objectives for this unit deal with working with multiple forms in a project. The objectives are know how to,

1. Use design and run-time properties to manipulate combo boxes and list boxes.
2. Use one/multi dimensional array.
3. Work with control and variable arrays.
4. Use message box and input box.
5. Work with fixed and variable length string variable.

4.1 Introduction

To simplify the task of writing programs, Microsoft has built in to VB a software program to help you write your VB projects. That software, known as the Integrated Development Environment (IDE for short) is what comes to the screen when you start VB and is the topic of this chapter. Visual Basic forms are windows. This chapter also provides details about how forms are handled in VB. Programming in VB is a combination of visually arranging components or controls on a form, specifying attributes and actions of those components, and writing additional lines of code for more functionality. To create professional applications, you need extensive knowledge of Visual Basic controls and their numerous properties, methods, and events. This chapter discusses some of the frequently used Visual Basic controls.

4.2 WORKING WITH MULTIPLE FORMS

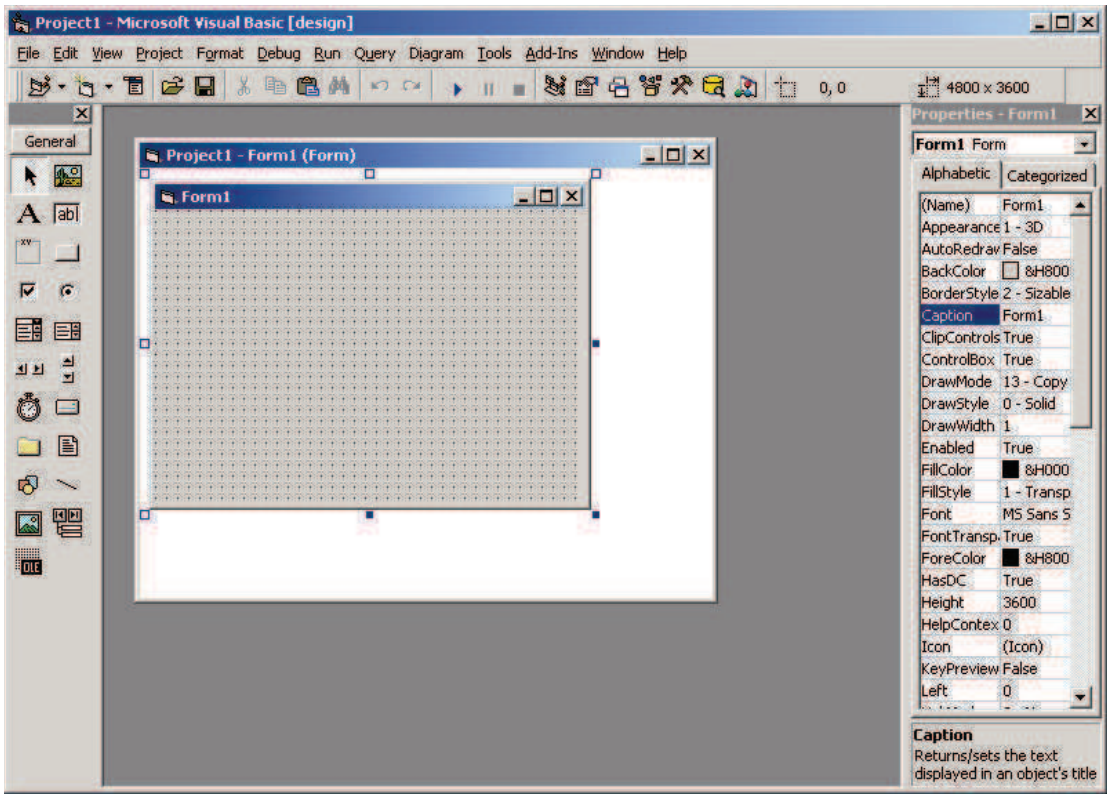
In Visual Basic, the basic building block of an application is a form, which is simply a window. A VB form looks just like any other form that you use in Windows applications. The header area has a caption, the control menu, and the minimize/maximize/close buttons. The large area of the form is called the client area. The VB IDE can insert forms into your project, and then you can resize the forms as well as change other properties of the form.

Although some programs you write will be simple so that you can use a single form in them, but due to limited amount of space on single form most will be made up of multiple forms.

Also by multiple forms you can design separate form for separate program tasks. For example, if you have a task in your program that is not performed often putting it onto a separate form makes more sense than trying to squeeze it on to a single form with every thing else. Also by loading and unloading form as you need then, you can save system resources. In other words your program takes up as little space as possible while running. This is done using the Multiple Document Interface (MDI) feature of Visual Basic. An MDI form is always contained within a parent form. This is exactly the same type of parent/child relationship which you see in Word. Each new Word document is contained in its own window, but is always framed within the larger window that is the Word application.

4.2.1 Adding new form

Visual basic starts a new project with a single blank form, as shown in the following figure.


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
4800 x 3600
General
Project1 - Form1 (Form)
Form1
Properties - Form1
Form1 Form
Alphabetic
Categorized

(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H800
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	■ &H000
FillStyle	1 - Transp
Font	MS Sans S
FontTransp.	True
ForeColor	■ &H800
HasDC	True
Height	3600
HelpContex	0
Icon	(Icon)
KeyPreview	False
Left	0

Caption
Returns/sets the text displayed in an object's title

Now you can add controls to this form and write code to handle events that occur on the form. At some point in your design, you may decide that you need one or more additional forms to handle a new task or provide space to relieve the crowding on the initial form.

For adding new form you can either click the add form button or select project®add Form. This action places a new black form on the screen. This form looks just like your first form. If you did not rename your first form from the default of form1 the new form is named from 2 (or Form 3, Form 4, and so on). Otherwise, the new form is named Form 1. After you add a new form, you can place control on it and write code for its events, just like for the initial form. You can also access new form from other forms in your program by following Statements (Load, unload) and methods (show, hide).

Table 4.1: Form Methods

Method/ Statement	Purpose	Syntax	Example
Load	Places a form in memory but does not display it.	Load form-name	Load form1
Unload	To remove form both from screen and memory.	UnLoad form-name	UnLoad form1
Show	To display form if form was not loaded. This method implicitly loads it and then displays it.	Form-name.Show	Forml.show
Hide	Removes the form from the screen but does not remove it from memory.	Form-name. Hide	Forml.Hide

Show method also has an optional argument that determines whether the form is shown as a modal or modeless form. If a form is shown modally (a modal form) then program control does not return to procedure that called the *show* method. Until the modal form is closed (example of modal form is windows 95 shut down screen) you cannot focus on another window while modal form is displaced, where as in modeless you can move to another form. To create modal form you can give show method as shown here.

Form1. Show vbModal

4.2.2 Adding Code module to project

As you write more code to handle more events and more tasks, you may often find that you need to access the same procedure from a number of different places on a form or from multiple forms. If this is the case, storing the procedure in a module file makes sense. To add module file to hold your procedure, you can do so either by clicking arrow on add form button and choosing module from drop-down menu or by choosing project®add module after clicking. The following screen will then be displayed.

Visual Basic gives the default name module1 to module. If you want you can change this name by changing value of name property of module.

4.2.3 Accessing the Forms and modules of a project

As you add forms and modules to program, they are added to project explorer window. You simply select a form or module by clicking its name in project window. For Form you will have two buttons.

View Object - to work on design of form.

View Code - to edit code associated with the form.

Whereas, for module only view code is enabled because a module has no visual elements.

Double - clicking name of form and module has same effect as clicking view object button and view code button respectively.

You can also access properties of controls of a Form in code of another Form By specifying form name, such as form name.control properties.

4.2.4 Variables and Constants in multiple forms project

Normally forms are complete in themselves. But in many programs, need of communication between forms is required. Imperatives regarding access to variables/constants used in multiple form projects are,

- Variable/constants declared in procedure of form can be used only in that procedure of that form (Private).
- If a variable is declared in General Section of Form code then it can be accessed by all procedures of that form.(Global to Form module)
- If variable is declared in General Section of module file without Public keyword then it can be accessed by all procedures of module file only.
- If a variable is declared by public keyword in module file then it can be used by all procedures in all form module and Procedures of module file also.

Self Learning Exercise-1

i. Fill in the blanks

a) Load method of form places a form in …….. but does not …….. it.

b) Module files have extension ……..

c) If a variable is declared by ……… keyword in module file then it can be used by all procedures in all form modules and Procedures of module file also.

d) Default access modifier for the methods in code window is ………

ii. True/False

a) In modeless form you can move to another form.

b) In modal form you can focus on another window

c) If a variable is declared in General Section of Form code then it can be accessed by all procedures of that form.

d) Hide Method of form removes the form from the screen and also from memory.

4.3 Use of List Box and Combo Box Controls

To the user, list box is similar to choosing channel on a TV. The cable company decides which channel to put on selection list, the customers then, can pick any of these channels but can't add one to the list if they don't like any of the choices provided. With the list box, choices are set up by the programmer, user can select only from the items the programmer decides should be available.

Both controls provide a list of choices but in different ways. A list box is limited to a list of choices. Form this list, user can make one on several choices. A Combo box enables the user to make one choice or to enter a choice. It is called a Combo Box because it is a combination of list box and text box.

4.3.1 List Box Properties :

Few commonly used properties of list box are,

List: This property can be used to add items at design time. After inserting an element, the user should press ctrl+Enter to insert new element.

Multi selection: allows user to make more than one selection from a list. This property can have three values 0, 1, 2 each one has following meanings -

0 (Default) Multiple selection not allowed

1 Simple multiple selection. This is done by clicking the mouse or pressing the space bar to select or deselect an item in a list.

2 Extended multiple selection (Shift+Click or shift + Arrow key) Extends selection to allow a block of items to be selected ctrl + click selects or deselects an item from the list.

Columns: Determines list box containing more than one column and how scroll bars are shown if list box needs to be extended. If value of this property is zero, list box is arranged in single column and vertical scroll bar. If value of this property is a positive integer then items arranged are in series of columns, filling each column from left to right and horizontal scroll bar.

List count: Represents number of items in a list box.

List Index: Represents the selected items. ((Note: -1 If no item is selected)

Text: Contains the item from whichever line of the list box that has the focus if no item is selected then this will have empty string (" ")

Sorted: To sort the list.

4.3.2 Combo Box Style Properties:

Most properties/methods of list box are also available in Combo Box. But Combo Box does not support multiple-selection. Combo Box are available in three styles which are as follows:

Drop Down Combo Box: Presents the users with a text Box combined with a drop down list. The users can either select an item from the list or type an item in the text box portion.

Simple Combo Box: Displays a text box and a list that does not drop down. As with drop down combo box user can either select an item from list or type an item in text box portion.

Drop Down List: Displays a drop down list box from which users can make a choice. The users can not enter items that are not in the list.

4.3.3 Methods associated with list box and combo box

Methods associated with list Box and combo box are given below

AddItem: To add item in list box at run time.

Syntax : <listbox name>. AddItem "<item"> [Index]

Example: List1.AddItem "First".

Index is used so insert an item at specified position.

RemoveItem: To remove an item from list, this method can be used. To specify an item to be deleted, index value can be used.

Example: List1. RemoveItem 0

Clear: To remove all items from list box.

Example: List1.Clear

The Load event occurs when a form is loaded in memory. This event can be used to fill combo and list boxes with data at runtime. As an example, examine the following procedure,

```
                 Private Sub     Form - Load ()
List1.AddItem   "Books"
List1.AddItem          "Magazine"
List1.AddItem          "Newspaper"
End sub
```

4.3.4 Using Run-time properties to manipulate list or combo Boxes

Not all properties of list or Combo Box controls are available to you at design time. Some of them are only available to you at run-time, and these are known as run-time properties. Some of these run-time properties are read only. That means only property's value can be retrieved during run time. You can't assign it a value with code. Runtime properties can be both retrieved and assigned with code. They are therefore known as Read-Write properties.

Some run-time properties are given below,

ListCount (Read only): Returns no of items in list or combo box. Its value changes as items are added or removed from list

Example Print List1.ListCount

List (read+write): You can retrieve or assign an item from/to list

Example List1.List (0) ="Report"

Print List1.List(3)

ListIndex (read/write): Indicate index value of the currently selected item in list/combo box

If you want to display an item in Combo Box, then you can assign index value of that item to ListIndex property.

Combo1.ListIndex=0 OR set 1st value as a default selection

Combo1.ListIndex = Combo1.ListCount – 1 OR set last value as selected item.

For example if you want to display items of a list,

```
For i = 0 to List1.ListCount-1
  Print List1.List (i)
Next i
```

Self Learning Exercise-2

i. Fill in the blanks

e) A ……….. combines the features of textbox control and a list box.

f) …………... method can be used to remove all items from list box.

g) ……….. property indicates index value of the currently selected item in list/combo box.

h) ……….. method can be used to add an item in list/ combo box.

ii. True/False

e) Multiple Selection is allowed in Combo Box.

f) Value of List Index property will be -1 If no item is selected

g) Maximum possible value for style property of combo box is four.

h) ListCount is ReadOnly Property.

4.4 MsgBox and InputBox Functions

The InputBox() and MsgBox() functions facilitate input and output of messages in the interface windows.

4.4.1 MsgBox ( ) Function

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he /she can continues. This message box format is as follows,

yourMsg = MsgBox(Prompt, Style Value, Title)

Table 4.2 Style Values

Button Layout	Style Value	Short Description
vbOKonly	0	Displays the Ok button.
vbOKCancel	1	Displays the Ok and Cancel button.
vbAbortRetryIgnore	2	Displays the Abort , Retry , Ignore
vbYesNoCancel	3	Displays Yes, No and Cancel button
vbYesNo	4	Displays the Yes / No button
vbRetryCancel	5	Displays the Retry and Cancel buttons.

Example:

```
yourMsg=MsgBox( "Click OK to Proceed", 1, "Startup Menu")
                        OR
yourMsg=Msg("Click OK to Proceed". vbOkCancel,"Startup Menu")
```

Table 4.3 Return Values and Command Buttons

Return Value	Value	Short Description
vbOk	1	The User Clicked OK
vbCancel	2	The User Clicked Cancel
vbAbort	3	The User Clicked Abort
vbRetry	4	The User Clicked Retry
vbIgnore	5	The User Clicked Ignore
VbYes	6	The User Clicked Yes
VbNo	7	The User Clicked No

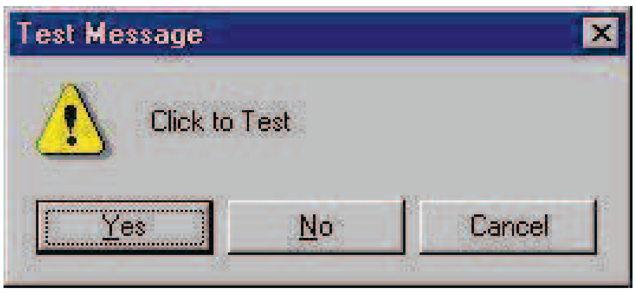
To make the message box looks more sophisticated, you can add an icon besides the message. The Icons displayed in the message box are described in table 4.4.

Table 4.4 Icon Descriptions

Icon on message	Value	Short Description	Icon
vbCritical	16	Displays critical message icon	❌
vbQuestion	32	Displays question icon	❓
vbExclamation	48	Displays exclamation icon	⚠
vbInformation	64	Displays information icon	ℹ

Example:

testMsg2 = MsgBox("Click to Test", vbYesNoCancel + vbExclamation, "Test Message")


Test Message

Click to Test

Message Box

4.4.2 The InputBox( ) Function

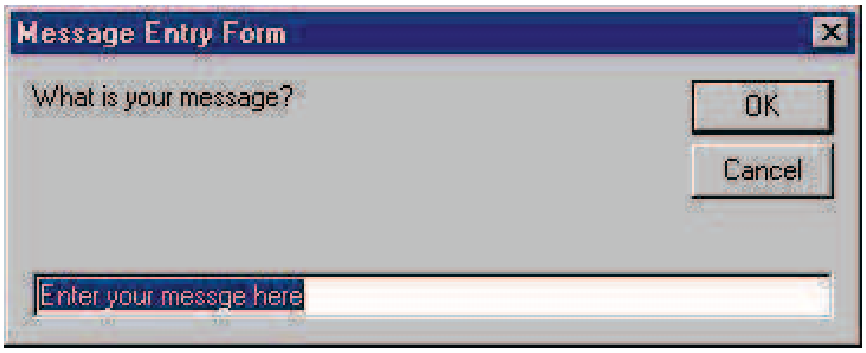
An InputBox( ) function will display a message box where the user can enter a value or a message in the form of text. The format is

myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)

myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

● Prompt	-	The message displayed normally as a question asked.
● Title	-	The title of the Input Box.
● default-text	-	The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.
● x-position and y	-	Position - the position or the coordinate of the input box.

userMsg = InputBox("What is your message?", "Message Entry Form", "Enter your message here", 500, 700)



Input Box

4.5 String Functions

Several built-in string functions perform string manipulations to augment simple concatenation with the "&" operator.

String variables can be assigned a series of characters or a null string. The value of a string can be assigned to another string variable.

A null string assignment is as given below

Person = ""

A text box on label is cleared with a null string as given below

Text1.Text = "" Labell.Caption = ""

Visual Basic allows you single concatenation operation that is ampersand symbol (&).

Consider the code,

```
First       =    "Sudha"
Second      =    "Morwal"
Myname      =    First & " " & second
Print Myname
```

This leads to the display

Sudha Morwal

Visual Basic provides several functions to manipulate string. Some of these functions are summarized in the following table,

Table 4.5 Built-in string functions

Use	Function	Example	Output
Length of the String	Len	instr = "Sudha" Result = Len (instr)	5

Change Case of string	Ucase
	Lcase
		Ucase("sudha")
		Lcase("SUDHA")
			SUDHA
			sudha

Searching a string Whether a word, phrase or group of characters exists in string. If so, where?	Instr (source-string, search-string) InstrRev(source-string, search-string) InstrRev Searches from last position	Print Instr ("I'll see you next Tuesday", "You")	10
Extracting pieces of string	Left – Retrieves specified no. of characters from left end of string. e.g. Left (Input string, number of chars) Right = Retrieves specified no. of chararcters from right end of string. Mid = Retieves characters from the middle of a string.	Print MID("Sudha Morwal",8)	Morwal

Fixed - Length Strings: Most strings that you use in your programs will be of the type known as variable - length strings. These strings can contain any amount of text. As information is stored in the variable, size of variable adjusts to accommodate the length of the string. Both implicit and explicit declarations shown earlier create variable length strings.

Visual Basic also provides strings of fixed length. As the name suggests fixed - length strings retain the same size, irrespective of the length of data assigned to it. If length of string variable is larger than the string assigned to it then remaining length of the variable is filled with spaces. If length of sting is shorter then the value assigned to it, the rest of the characters will be truncated. Only as many characters as will fit in the variable are stored.

A fixed length string variable can only be declared using on explicit declaration like the following.

Dim variable-name As Strings *string-length

For example

Dim sName as string * 25.

Self Learning Exercise-3

i. Fill in the blanks

i) Output of Print Instr ("I'll see you next Tuesday", "see") is ……..

j) Output of Len("Sudha") is ………

k) A null string can be represent by ………….

ii. True/False

i) Print Ucase(“softech”) displays Softech.

j) MID(“computer”,1,4) gives output comp

k) String variable can be assigned null string.

l) Symbol (+) is used to concatenate two string

4.6 Arrays

Generally one variable can hold a single value only. Therefore if we want to store 10 values, using 10 variables can be a solution. But it is not a good practice to use so many variables, therefore, in this situation, we require a single variable which can store multiple values. Such a variable is called an Array. So Array is collection of values which is referred by a single name. These values should be of same type. In memory, an array occupies consecutive memory locations so all values are stored consecutively. To declare a variable of Array data type, we can use.

Dim <Name of variable> (size)

for example, Dim *ar* (5)

In above example *ar* is an array of 6 values. Array index starts with zero and upper bound of index in ar(5) is 5. So we can store total 6 values in the array i.e. *ar*(o), *ar*(1), *ar*(2), *ar*(3), *ar*(4) *ar*(5). If data is not defined as in above example then it will be automatically of type variant and if you want to specify array of 5 integers then you can declared as given below,

Dim ar (5) As Integer.

As By default index is zero But you can also specified range of index value as given below :

Dim ar (1 to 10) As Integer.

Now index will start with 1 and 10 is last index.

LBound, UBound are the functions which determine lower bound and upper bound of an array. Use of these functions is given as follows,

```
Dim A(5) As Integer
Print UBound (ar)
          Print LBound (ar)
```

4.6.1 Types of Arrays

Visual Basic provides two types of Arrays Fixed-length and Dynamic arrays.

(i) Fixed-Length Arrays

In fixed length arrays, size of Array is constant. Scope of Array is defined as follows,

a. To create a **global array**, place public statement in the declaration section of standard code module to declare the array. No further declarations are needed.

b. To create a **module - level array,** place Dim Statement in the General declaration section of a form module or in standard module. No further declarations are needed.

c. To declare an **array with local scope**, declare within procedures.

To declared two dimensional Array we can use,

Dim ar (1 to 3, 1 to 5) as integer

(ii) Dynamic Array

Mostly we don't know size of an array in advance. So we have to change array size at run time. In that situation, we can declare dynamic array, size of which can be changed later on. To declare dynamic array, we can use,

Dim *ar* () as Integer.

To define its size we use ReDim statement as follows :

ReDim *ar* (4)

Redim can be used in procedure only.

Dynamic allocation is important for several reasons; first it allows the size of an array to be increased (on decreased) as items are added (on deleted) from a list. In many programs, the size of list will not be known in advance. ReDim allows the size of a list to be kept flexible.

Second, Redim allows an array to be declared at procedure level. As a rule, define all variable, including array variables at the lowest level possible. Redim provides this capability.

Third, it is possible to release the memory allocated for an array. By reducing the size of an array to zero, all memory is released as in the instruction,

Redim *ar* (0)

As we redeclare array by ReDim, Visual Basic stores default value in memory allocated dynamically. So values stored earlier in that array will be lost. If you want that earlier values should be retained by array you can use Preserve keyword in ReDim declaration as given below-

Redim Preserve *ar* (size)

But by Preserve keyword only upper bound can be changed. We can't change data type of an array using ReDim if array is of data type other then variant type. But data typed of array of variant type can be changed. Example is given below

```
Dim ar () as integer
Dim br (5) as integer
Private Sub Form1 _Activate ()
        Redim ar (4)
        Print Ubound (ar)
        i = Ubound (br)
        Redim preserve br (i+5)
        Print Ubound (br)
End Sub
```

Output will be

```
4
10
```

4.6.2 Control Array

Control array is a group of controls, all of same type, that have the same name and are identified by an index. Control arrays are similar to array.

Advantages of control array are,

- Adding an element to a control array requires fewer system resources than adding an individual control of same type. For example three independent text boxes (Text1, Text 2, Text3) use more resources than a three element text-box array.
- The elements of control array share a common set of event procedures. This means you have to write program code in only one place to handle the same event for all controls in the array.
- Control arrays provide the only means of adding controls to a form while your program is running.
- Using control array, you can avoid hitting the limit of 254 control names per form. This allows you to add as may controls as you need.

Creating Control Array

You must create a control array at design time. Although you can add elements to a control array at runtime, at least first element of array must be created in design environment.

You can create control array using following methods,

- Click to control and select edit and copy. Click on form and select edit & paste. So new ctrl. is shown in same position as the original control because it inherit all the properties of original control. So drag new control to the position where you like to place it.

OR

- Add individual controls to form and then change the name properties of all controls to the same value. After these step, a dialog box appears, press yes in that.

After you create control array, you can write a single code to handle a particular event for all controls in an array. For each event Visual Basic passes the index property of controls that triggered the event to the event procedure as an argument.

To write code for each of the elements of a control array, you simply enter the code in code window just as you would for any other control then whenever that event is fixed by any of the controls in the array, the code is executed.

Self Learning Exercise-4

i. Fill in the blanks

l) Collection of controls having same name and identified by index is known as …...

m) The **…………** Keyword allows you to resize an array without loosing content

n) The **………..** keyword allows you to resize an array.

ii. True/False

m) By Preserve keyword only upper bound can be changed.

n) Change of data type of an array using preserve can not be possible even in the case of array is of variant data type.

o) Redim allows an array to be declared at procedure level.

p) Using control array you can avoid hitting the limit of 254 control names per form.

4.7 Form Initializing & Terminating Events

Load	-	Occurs when a form is loaded into memory. You can use this event to fill combo and list boxes with data.

Unload	-	Occurs when a form is removed from memory.
Activate	-	Occurs as each new instance of the form is created before the load event occurs. Also occurs when the user returns to the form from other forms.
Got Focus	-	When an object (Form/control) receives Focus, this event occurs.
Lost Focus	-	It's the LostFocus event when, focus has left the object. Special events occurs on each form you can place code in. We associate event procedure for any of them.
Deactivate	-	Occurs when user moves to another form or form is hidden.
Initialize	-	Occurs when an instance of form object is created.
Terminate	-	Occurs when an instance of form object is destroyed.

4.8 Summary

This chapter has covered several topics critical to the study of VB. It discussed how you can use InputBox and MsgBox functions in program to accept value and print any value respectively.

Control arrays were introduced. A control array is nothing more than a group of objects with the same name. VB provides the index number for each object within the group. For example you might have a control array of text boxes and another control array of list boxes.

This chapter also discussed list and combo boxes as examples of one dimensional lists of items. With combo box determining which item has been selected is much easier because this item always appears in the combo box.

This chapter also described the use of one dimensional variable arrays. A one dimensional array is a list of more than one variable with the same. Variable arrays can be declared in several ways. The preferred one is, ReDim ar(5) as string. ReDim permits the size of an array to be dynamically controlled, keeps the variable declaration at procedure level, and permits memory to be released by reducing the size of array to zero.

Array subscript always begins with zero.

To ensure that previously assigned values to an array are not lost when array is redimensioned, use preserve keyword.

String is a sequence of characters. String functions deal with ways of manipulating and converting strings. Set of string functions deals with converting a string from one form to another.

Left (), Right () and Mid () return part of String.

RTrim (), Trim () and LTrim () remove blank spaces.

Instr () returns position of sub-string within a larger string.

This chapter also discussed How you can develop a project having multiple forms. Load function is used to load form in memory, Show and Hide functions are used to display or hide a form from a screen.

4.9 Glossary

IDE	IDE means a software in which there is an environment provided to the user in which there are so many tools available for coding, testing, debugging, deployment and maintenance.
Control Array	Control array is a group of controls, all of same type, that have the same name and are identified by an index

4.10 Further Readings

1. Alan Eliason, Ray Malarkey, "Visual Basic 6, Environment, programming and Application".
2. Brain Siller and Jeff Spotts, "Visual Basic 6".

4.11 Answers to Self Learning Exercises

i. Fill in the Blanks	ii. True/False
a) Memory, Display	a) True
b) .BAS	b) False
c) Public	c) True
d) Private	d) False
e) Combo Box	e) False
f) Clear	f) True
g) ListIndex	g) False
h) AddItem	h) True
i) 7	i) False
j) 5	j) True
k) ""	k) True
l) Control Array	l) True
m) Preserve	m) True
n) ReDim	n) False
	o) True
	p) True

4.12 Unit End Questions

1. Write a VB program which work as a calculator. (Use control array for digits)
2. Design a VB application which has two list boxes one of them has MultiSelect property TRUE. Add Two buttons one of them transfer the selected items from First list box to second list box and other will transfer the selected items from Second list box to First list box.

Unit - 5: Working With Files

Structure of the Unit

5.0 Objectives

At the end of this unit you will be able to,

- Use built-in VB file processing system control.
- Understand concepts important to design of Sequential File records and Random Access files.
- Access data base file.
- Apply connectivity through data control.
- Understand three methods of database connectivity DAO,ADO and RDO

5.1 Introduction

If data needed by a program are to be saved the data must be stored on disk, a computer can store two types of files; program files & data files. Program files contain VB or application specific instructions. These include .VBP, .FRM, .BAS files. Data files contain the data to read to and from program files. These files will include text files(.txt) and data files(.dat).

This chapter begins with an introduction to four file processing controls - drive list box, the directory list box, the file list box and common dialog control. Following this, we examine how to write to and read from a sequential file besides sequential file processing VB also provide random access file processing(default). With random access file processing, fixed length records are managed and all records must correspond to one type. VB includes some special controls (ADO, DAO, RDO) for accessing external databases. These controls allow a user to access an existing database and to navigate through the collection of stored data with little or no programming.

Built-in Visual Basic File processing system Control

Many Visual Basic applications require information about the current disk drive (which is open), the directories stored on the disk placed in the current drive, and the file stored within a directory or placed on the disk. Visual Basic provides four controls to avail facility that is just described.

Drive list Box displays available disk drive on a computer.

Dir list box displays the directories created for a disk placed in disk drive.

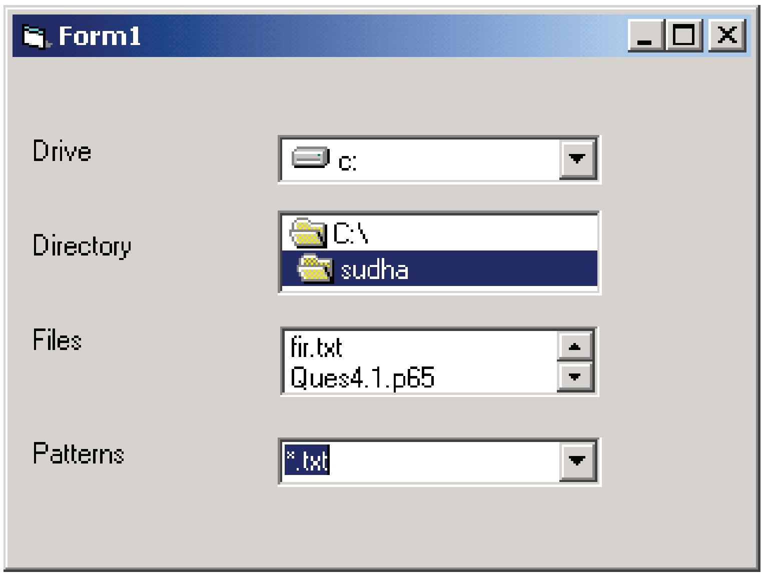
File list box lisplays files stored within the directories including root directories.

Collectively these three tools tells you what files are stored on a specific drive within the named directory

Common Dialog Control

Opens standard dialog boxes such as open, color, print and save dialog boxes. If common dialog control is not available on tool box the add it by selecting component option of project menu, after that select Microsoft Common Dialog Control 6.0.

Example



Step1 : Drag and drop drive, directory, file list boxes on form

Step2 : Write following code in change event of drive list box

```
Private Sub drive_change()
        dir1.path = drive1.drive
End Sub
```

Step3 : Write following code in change event of dir list box

```
Private Sub dir_change()
        file1.path = dir1.drive
End Sub
```

Step 4 : Take a combo box on form and add following items in list property of that combo box.

```
*.txt
*.doc
*.*
```

Step 5 : Add the following statement in change event of combo box

file1.pattern = combo1.text

Step 6 : Save and Run

5.2 Data Files

Data files contain the data to read to and from program files. Database file is an example of data file. We will describe database file in detail later on.

5.2.1 Sequential File Organization

Sequential means that the file is accessed one byte after the other in sequence, rather than jumping to a specific location.

Sequential access is especially useful for text files, where each character in the file represents either a character (including blank space) or a text formatting characters (newline). Text formatting is limited to reading and writing of strings of text, rather than a set of records (unit of data to be processed by computer at a time is known as record). With sequential access, two files are typically required - one to read data from and one to write data to. It is not possible to read and write to same file at the same time. If, for example, you want to insert a record into a sequential file, it is necessary to read from one file and write to second file until the desired location is reached. At this time a new records is inserted and written to second file (not the first). Once this is done, all remaining record must be read from first file and written to second.

Functions and statements

Most useful functions and statements applied to sequential file are as follows -

EOF	Returns a value to indicate whether the end-of-file has been reached.
LOC	Returns the current position within an open file.
LOF	Returns size of an open file in bytes.
Seek	Sets the position in a file for the next read or write.
Open	Enables input or output to a file.
Close	Closes input/output to a file.
Input #	Reads characters from a sequential file and assigns data to variables.
Print #	Writes formatted data to sequential file.
Write #	Writes raw data to a sequential file.

How these functions and statements can be used is described below-

Open

Syntax Open file For [mode][Access] [lock] As [#filenumber [len = reclen]]

Example Open "notes.txt" For Output As #1 (if file does not exist then it will create it.)

Write

Syntax Write# filenumber [, expression list]

Example Write #1, a,b,c (this will write values of variables a, b and c in the file having number 1)

Print

Syntax Print file number [[{spc(n)|tab(n)}][expression list][{;|,}]

Example Print #1, Text1.Text

Input

Reads list of numeric or string expressions from a file. This statement is used when the exact type of data is known in advance.

Syntax Input #filenumber, variable list

Example Input #1, a,b,c (three values read from file which have file number1 and store in a,b,c respectively)

LineInput

This statement reads a text file one line at a time, reading all characters up to a carriage return.

Syntax LineInput #file number, variable name

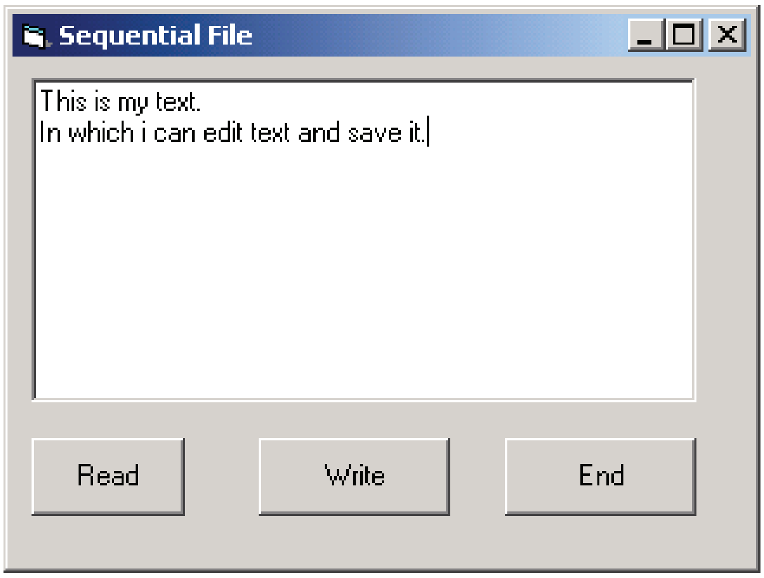
Example LineInput #1, textline

Close

Syntax	Close [[#]filenumber][,[#]filenumber]...
Example	Close #1

Example Program:

This exercise accepts a text and stores and retrieves it in/from text file, as shown on following screen.



This is my text.
In which i can edit text and save it.

```
Private Sub write_click()
Open “notes.txt” For Output As #1
Write #1, Text1.text
Close #1
Text1.text = “ “
read.enabled = true
write.enabled=false
End Sub
```

```
Private Sub read_click()
  Open “notes.txt” For Intput As #2
  Do Until EOF(2)
    Input #2, linetext
  Text1.text = Text1.text & linetext
  Loop
  Close #2
End Sub
```

Kill

To delete a file from a disk

Syntax	Kill filespec

filespec is name of file. Kill will not work if file is open.

Self Learning Exercise-1

i. Fill in the blanks

a) Kill is used to .... a file from a disk

b) EOF returns a value to indicate whether the .... has been reached.

c) Print # writes .... data to sequential file.

d) Write # writes ……….. to a sequential file.

ii. True/False

a) Sequential access file means that the file is accessed one byte after the other in sequence, rather than jumping to a specific location.

b) Data files contain the data to read to and from program files.

c) Database file is not an example of data file

d) In sequential files it is possible to read and write to same file at the same time.

5.2.2 Random Access Files

Sequential files do not have any structure. As a matter of fact, the structure is defined by the code that read the file rather than in the file itself that's why Visual Basic does not contain a function to jump to a specific record in a sequential file

One way to move some structure into the file itself is to store user defined record rather than just string data. You can then open in Random mode and are not limited to sequential access.

Creating Record Type

You can use user-defined data type by using Type statement to create a custom record. Type declarations are entered in general section of code window.

For example following code declares a student record type

```
Private Type student
        roll-no As Integer
        name As String*30
        percentage As Single
End Type
```

Custom record type can be accessed from code as follows-

```
Dim std As student
std.name = "sudha"
std.roll-no = 15
std.perc = 70.98
```

Functions and Statements

Seek

Moves file pointer to specific record number.

Syntax	Seek filenumber, record number
Example	Seek #2, 10 (now next record to be read or write will be 10th recording file #2)

LOC

Returns current location in a file.

Syntax	LOC(file number)
Example	LOC(#2) (return 9, since10 is next record to process)

Open

Opens a file.

Syntax	Open file for Random As file number len = record length
Example	Open "test.dat" For Random As #1 Len = RecordLen ' where RecordLen = Len(rec-name)

Close

Same as in sequential access file

Get

Reads from a file into variable

Syntax	Get [#] filenumber,[record number], variable name
Example	Get #2, 10 (This read 10th record)

Put

Writes from a variable to a disk file

Syntax	Put [#] file number, [record number], variable name
Example	Put #1, std (if no record number is specified then record will be written to current file pointer)

Seek

Moves from record to record.

Syntax	Seek file number, record number
Example	Seek #1,3 (next Get and Put will be 3rd record)

Example

Following code accepts five records from user and writes to file #1.

```
For i = 1 to 5
        std.roll-no = InputBox("Enter roll number")
        std.name = InputBox("Enter name")
        std.perc = Val (InputBox("Enter percentage")
        Put #1, std
Next I
```

Following code reads 4th record of file #1 and prints it.

```
Get #1, 4, std
MsgBox "name = "&std.name
MsgBox "roll number = "& std.roll-no
MsgBox "Percentage = "&std.perc
```

Visual Basic also provides third type of file that is Binary file which allows variable length record size. Whereas Random file allows only fixed length record size. If for example, a student's name is allocated 30 bytes but only requires 10, then 20 bytes of storage would be wasted with random access storage. With Binary storage only characters entered are stored. If student's name requires 10 bytes then only 10 bytes are used.

Self Learning Exercise-2

i. Fill in the blanks

e) **Seek** moves ……….. to specific record number.

f) **LOC** returns current ………… in a file.

g) **Get** reads from a …….. into variable.

ii. True/False

e) Random files do not have any structure.

f) Binary file allows variable length record size.

g) Random file allows fixed length record size.

5.3 Accessing Database Files

Visual Basic has been a tool of choice for database programmers everywhere. With each successive version, Microsoft has added more functionalities to make database programming and hence access easier. VB's powerful database feature set has grown with the addition of new tools and technologies with new versions.

5.3.1 Terminology

Before going in detail of data base access facility of VB we have to know some terms regarding database. Some of them are given below,

Data	Data are known as facts.
Database	Collection of data.
DBMS	Collection and management of data is known as **D**ata **B**ase **M**anagement **S**ystem.
RDBMS	(Relational DBMS) is a system in which many related databases can be managed.
Table	A table is a logical structure used to group sets of related information. For example one table represents information about a set of students (roll-no, name, address) and another represents marks (roll-no, mark-obtains, total, percentage).
Field	A field is an attribute of a record. An attribute of student database is roll-no, another is name.
Primary Key	Attribute which can uniquely identify a record. For example in student database roll-no of each student will have unique value so it can uniquely identify each record therefore we can set roll-no a primary key.
Foreign Key	Foreign keys are fields in common, between tables. For example we have roll-no common in both tables (student, marks). So roll-no is primary key in student and foreign key in marks table.
Indexes	Access to a database record is often made faster through use of an index assigned to fields rather than primary key field. Suppose you want to retrieve a student record by entering name of the student, rather than the unique roll-no. To accomplish this you can create an index based on student name keeping this index separate from the data base table. Before retrieving a record from the table, the index is searched for record containing the specified student name. If the search is successful the record location of student is identified. This allows direct access to student record.
Queries	Queries are questions asked for database. In VB Structured Query Language (SQL) is used to retrieve data from a database. For example

```
SELECT  name,address
FROM    student
WHERE   roll-no = 15
```

5.3.2 Structure of database application

A Database application contains three elements

User Interface

Users interact with this component. This includes all screens used to view and modify data.

Database Engine

This is a collection of DLL which combines with programs at run time. This is responsible to read, write and modify database.

Data Store

A data store is a collection of files, database tables.

5.3.3 Database Connectivity through Data Control

Visual Basic is designed to enable a developer to create window-based applications quickly and easily. This extends to creation of database applications as well. Visual Basic enables you to write a complete data management application with no programming. The components that make these capabilities possible are data control and data bound control.

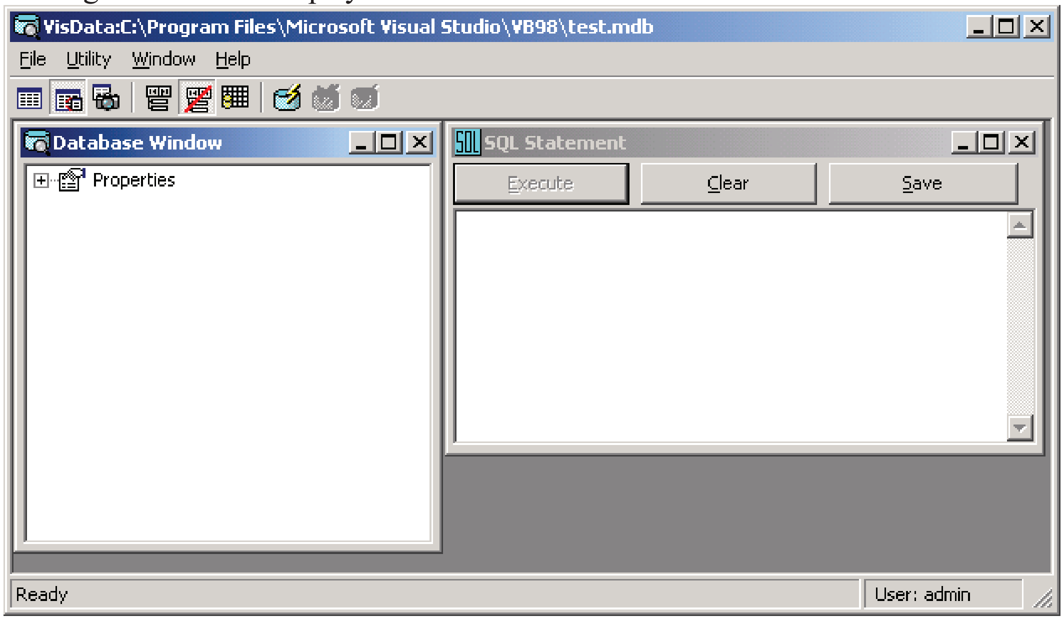
This is the easiest way to connect database by simply setting data control properties, it is possible to connect to database without writing a single line of code, through the Data Control.

Creation of database using database manager Application

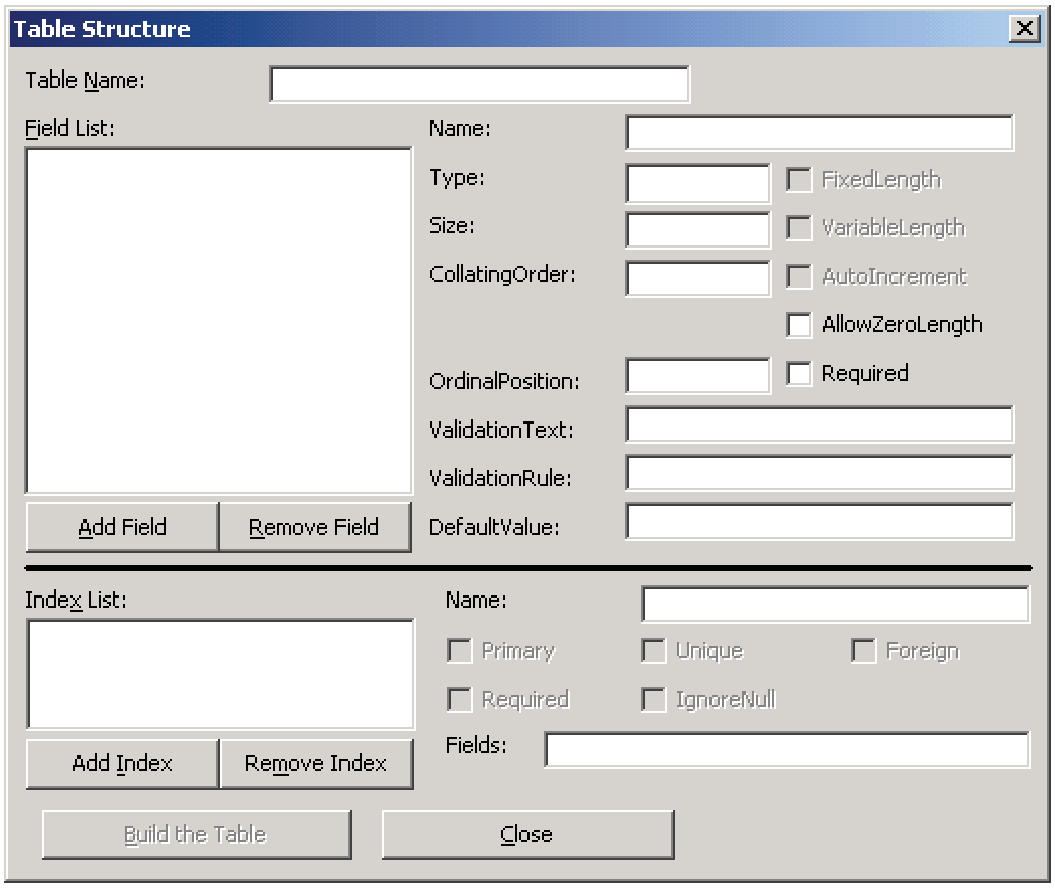
It is possible to create database within VB environment. For this follow the following steps-

Step 1. Select Visual Data Manager option of AddIns menu in VB environment.

Step 2. Select the database, say, Microsoft Access. Then enter the database file name. Following screen will be displayed

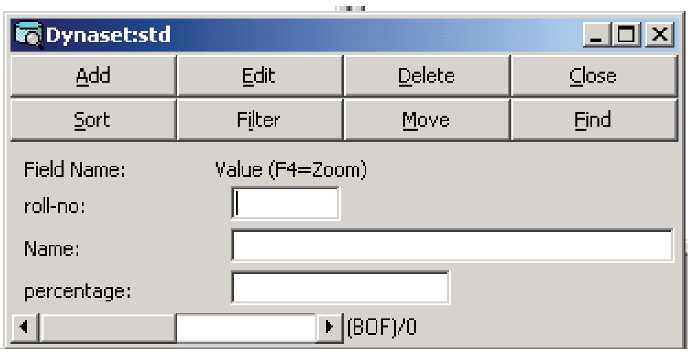

VisData:C:\Program Files\Microsoft Visual Studio\VB98\test.mdb

Step 3. Right click on properties and then select new table. Following screen will displayed



Step 4. Feed table name and then click on add field button. Enter all fields information(name, type, size etc.). Repeat by pressing Add Field button for as many fields as required. Then press Build the table.

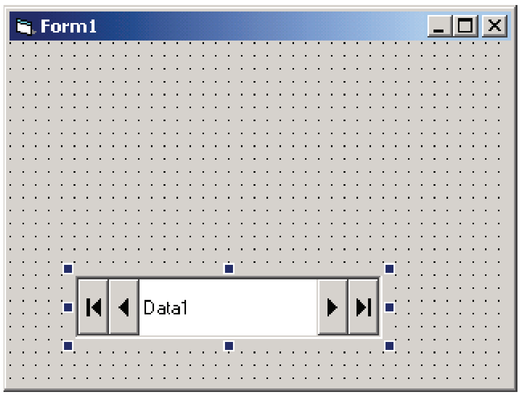
Step 5. To Enter the data in table right click on table name and select open, the following screen will be displayed.



Step 6. Enter as many data as you want by repeatedly clicking Add button.

How to connect a database using Data Control

Step 1. Open a new project in VB and add data control on form as shown on the following screen.

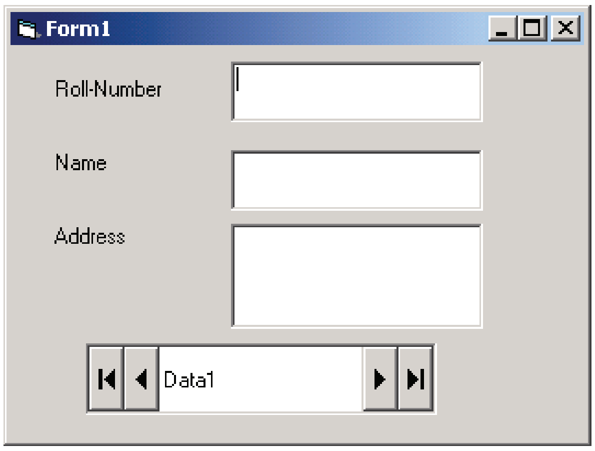


Step 2. Select Data Control(data1) and change following properties

DataBaseName - File name of database file

Record Source - Table Name/ Query Name

Step 3. Add Bound Control to display contents of records of table of selected database. As an example we add text boxes as a bound control. So add Text boxes to display data as given on the following screen.



Step 4. Select Text Box and change the following properties.

Data Source - Data Control name (say Data1)

Data Field - Field name of table selected in step 3 (say roll-no)

Step 5. Repeat step 4 for all fields you want to be displayed.

Step 6. Run the Application. First record will be displayed and you can go to next records using navigation buttons of data control.

One of the most frequently used properties of the data control is the Recordset.

Recordset

Three alternative values of this property are as follows,

Table	A data set that is returned from table of database and may be updated.
Dynaset	A data set that is returned from a query and may be updated.
Snapshot	A data set that may be either table or a query but is read only, that is, may not be updated
EOF / BOF	Indicate End of File(i.e. last record) and Begin of File(i.e. First Record). Use of

these properties will be clear by following example code (in this code, the programmer wants to prohibit users to go to the position after last record. Similarly, you can write code to prohibit user to go to position before the first record).

```
Private Sub move_next()
        If Not Data1.Recordset.EOF then
                Data1.Recordset.MoveNext
        else
                MsgBox "You are already at last record."
        end if
End Sub
```

While creating a database application you will need some more methods. Some of them are described below.

Refresh

To open database,

To reopen database with different set of data control properties,

To move to first record in a recordset.

Syntax	[form.]datacontrol.Refresh
Example	Data1.Refresh

AddNew

Clears the buffer in preparation for creating a new record in table or dynaset

Syntax	[form.]datacontrol.AddNew
Example	Data1.AddNew

Delete

To delete current record

Syntax	[form.]datacontrol.Delete
Example	Data1.Delete

Edit

To open current record for editing

Syntax	[form.]datacontrol.Edit
Example	Data1.Edit

Update

To save the contents of copy buffer to table of dynaset.

Syntax	[form.]datacontrol.Update
Example	Data1.Update

FindFirst

To locate the First record that satisfies specified criteria and makes it the current record.

Syntax	[form.]datacontrol.FindFirst criteria

Example Data1.FindFirst roll-no = ‘5’

FindNext

To locate the Next record that satisfies specified criteria and makes it the current record.

Syntax [form.]datacontrol.FindNext

Example Data1.FindNext

FindPrevious

To locate the Previous record that satisfies specified criteria and makes it the current record.

Syntax [form.]datacontrol.FindPrevious

Example Data1.FindPrevious

FindLast

To locate the Last record that satisfies specified criteria and makes it the current record.

Syntax [form.]datacontrol.FindLast criteria

Example Data1.FindLast roll-no = ‘5’

MoveFirst

To move to the first record in a specified recordset and make it the current record.

Syntax [.form]datacontrol.MoveFirst

Example Data1.MoveFirst

MoveNext

To move to the Next record in a specified recordset and make it the current record.

Syntax [.form]datacontrol.MoveNext

Example Data1.MoveNext

MovePrevious

To move to the Previous record in a specified recordset and make it the current record.

Syntax [.form]datacontrol.MovePrevious

Example Data1.MovePrevious

MoveLast

To move to the Last record in a specified recordset and make it the current record.

Syntax [.form]datacontrol.MoveLast

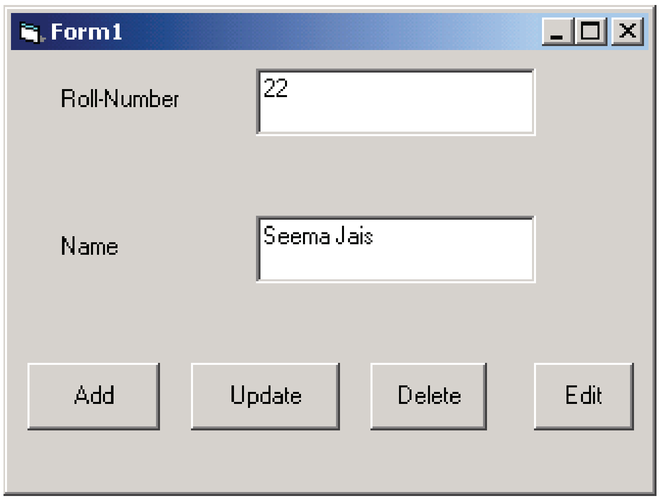
Example Data1.MoveLast

Seek

In table type recordset, moves to the indexed record specified by method argument.

Example:

This exercise assumes that a table with name “std” is created in chk.mdb database. Now design an interface to add, delete and modify records in table. Update button is used to indicate that record should now be saved. Following figure represents the interface and after that code corresponding to each button is also given.



```
Private Sub Add_Click()
        Data1.Recordset.AddNew
End Sub
Private Sub Update_Click()
        Data1.Recordset.Fields("roll-no") = Val(Text1.Text)
        Data1.Recordset.Fields("name") = Text2.Text
        Data1.Recordset.Update
End Sub
Private Sub Delete_Click()
        If Not Data1.Recordset.EOF Then
                Data1.Recordset.Delete
                Data1.Recordset.MoveNext
        Else
                MsgBox "No record to delete"
        End If
End Sub
Private Sub Edit_Click()
        Data1.Recordset.Edit
End Sub
Private Sub Form_Load()
        Data1.Refresh
        Text1.Text = Data1.Recordset.Fields("roll-no")
        Text2.Text = Data1.Recordset.Fields("name")
End Sub
```

In case of data in result of query. Design time binding will be as follows-

1. Write the query in the visual data manager window
2. It will ask name of the query.
3. Now give recordsettype - Snapshot and recordsource - Query name

Rest of the process is same.

You can also give query data at run time. Example code is given by-

```
data1.RecordSource  = "SELECT [name],[roll-no] FROM std"
data1.Refresh
```

5.3.4 Database Connectivity through Objects (DAO, ADO, RDO)

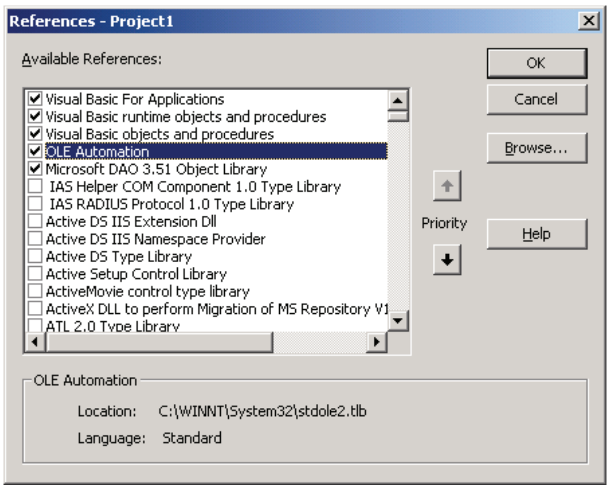
DAO (**D**ata **A**ccess **O**bjects)

Data Control provides limited access to database but there is no or less programming when it is used. But DAO is a powerful Object oriented programming interface which provides full control on database but there is more programming required. And also DAO help you to understand the concepts behind the controls. You can perform more detailed input validation before an attempt is made to enter the data into database.

This is also an efficient way to handle the database using DAO. Steps are given below,

Step 1. Take a new project

Step 2. Select references option of Project menu. Following window will displayed.



Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] Microsoft DAO 3.51 Object Library
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] Active DS IIS Extension Dll
- [ ] Active DS IIS Namespace Provider
- [ ] Active DS Type Library
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library
- [ ] ActiveX DLL to perform Migration of MS Repository V1
- [ ] ATL 2.0 Type Library

OLE Automation

Location: C:\WINNT\System32\stdole2.tlb

Language: Standard

Step 3. Select Microsoft DAO 3.51 Object Library. Press OK.

Step 4. Write following declaration in general section of code window.

```
Dim db As Database
```

Step 5. Write following statement in load event of form.

```
Set db = OpenDataBase("chk.mdb")   ' Path if required
```

Step 6. Write following statement in unload event of form.

```
db.close
```

Step 7. To access recordset object add following declaration in general section.

```
Dim rs As Recordset
```

Step 8. Add following statement to load event handler

```
Set rs = db.OpenRecordset("std",dbOpenTable)
```

```
        rs.MoveFirst
```

Step 9. Add controls(say text boxes) to display data to form. Add following code to load event

```
        text1.text = rs("roll-no")
        text2.text = rs("name")
```

Step 10. Add Two command buttons change caption and name of button to Next and Previous respectively.

Step 11. Add following code to click event of Next button

```
        Private Sub cmdNext_click()
            If  Not rs.EOF   Then
                rs.MoveNext
                Text1.text = rs("roll-no")
                Text2.text = rs("name")
            Else
                MsgBox "You are at the Last Record"
            End If
        End Sub
```

Step 12. Add following code to click event of Previous button

```
        Private Sub cmdPrevious_click()
            If  Not rs.BOF   Then
                Text1.text = rs("roll-no")
                Text2.text = rs("name")
                rs.MovePrevious
            Else
                MsgBox "You are at the Last Record"
            End If
        End Sub
```

Step 13. Test and Run the application.

In DAO programming Recordset properties can have four types of values; those are, Table type, Dynaset, Snapshot and Forward Only. First three are already described above.

Forward Only Recordset

It is same as Snapshot type but only allow forward scrolling through its record. This means MoveFirst, MovePrevious, Find methods do not work on the Recordset. It is faster then Snapshot.

To set type of Recordset any one statement of following statements can be used-

```
        Set rs = db.OpenRecordset("std",dbOpenTable)      ' std is the name of a table
        Set rs = db.OpenRecordset("std",dbOpenSnapshot)
        Set rs = db.OpenRecordset("std",dbOpenDynaset)
        Set rs = db.OpenRecordset("std",dbOpenForwardOnly)
```

Other methods (rs.AddNew, rs.Edit, rs.Delete, rs.MoveNext ect.) work as in data control.

RDO (Remote Data Object)

Visual Basic is a great tool for creating front end for client/ server application. These types of applications are used to access data stored in database servers such as SQL server and Oracle. Visual Basic program can easily access data stored in a variety of remote locations through use of RDO. After connection to data source is made, same code statements that are used for DAO can be used to access the data using RDO. To give a feel for similarities between RDO and DAO, following table lists a number of RDO objects and their corresponding DAO objects-

rdoEngine	dbEngine
rdoEnvironment	Workspace
rdoConnection	Database
rdoTable	TableDef
rdoResultSet	Recordset
rdoColumn	Fields
rdoQuery	Querydef

rdoResultset can have following values-

rdoResultset Type	Recordset Type	Definition
Keyset	Dynaset	Updatable set of records.
Static	Snapshot	Non-Updatable set of records.
Dynamic	Not Applicable	Similar to Keyset
Forward Only	Forward Only	Similar to static, but you can move forward only through set of records.

RDO do not support any rdoResultset Type that returns a table. This is because RDO are geared to using SQL statements to retrieve subset of information from one or more tables. Also because there is no table equivalent, RDO does not support indexes.

AddNew, Delete, Edit, Update and all Move methods are also available in RDO.

```
Dim db   As rdoConnection
Dim rs   As rdoResultset
Dim sSql As String
Set db = rdoEngine.rdoEnvironment(0).OpenConnection(“MyDSN”)
sSql  =  “SELECT * FROM std”
Set rs = db.OpenResultset(sSql, rdOpenKeyset)
rs.MoveFirst
While Not rs.EOF
        Print rs.rdoColumns(0)
        rs.MoveNext
Wend
rs.close
db.close
```

ADO(Active Data Object)

ADO is an object oriented interface which provides facility to access database using OLEDB data providers. ADO provides fast and easy access to data, ADO have mainly three objects-

Connection: Object of highest level. This establishes connection between application and data source.

Command: This object is used for querying data source.

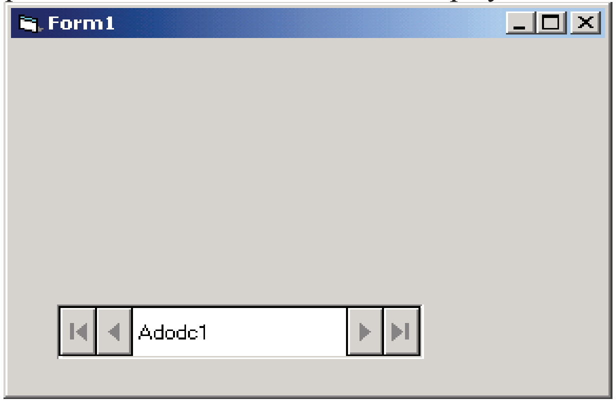
Recordset: This object is used to access records which are result of query. Using this object we can add, remove and modify records.

Data control is available for design time data connectivity corresponding to DAO. Similarly ADODC is also available for design time data connectivity corresponding to ADO. Here we will only discuss data connectivity through ADODC.

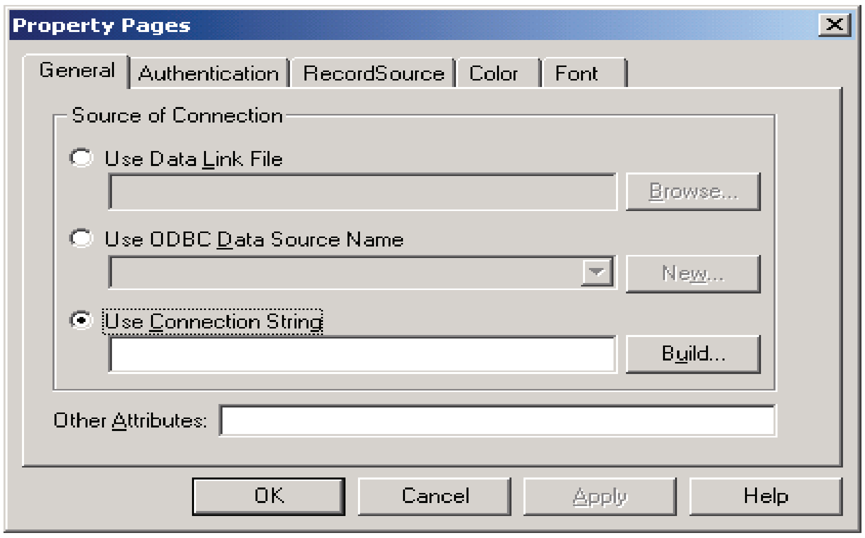
1) To add ADODC on tool box select components of project menu.

2) Select the check box Microsoft ADO Data Control 6.0 (OLEDB) then press OK.

3) Drag and Drop ADO Data Control on Form. as displayed in following screen.



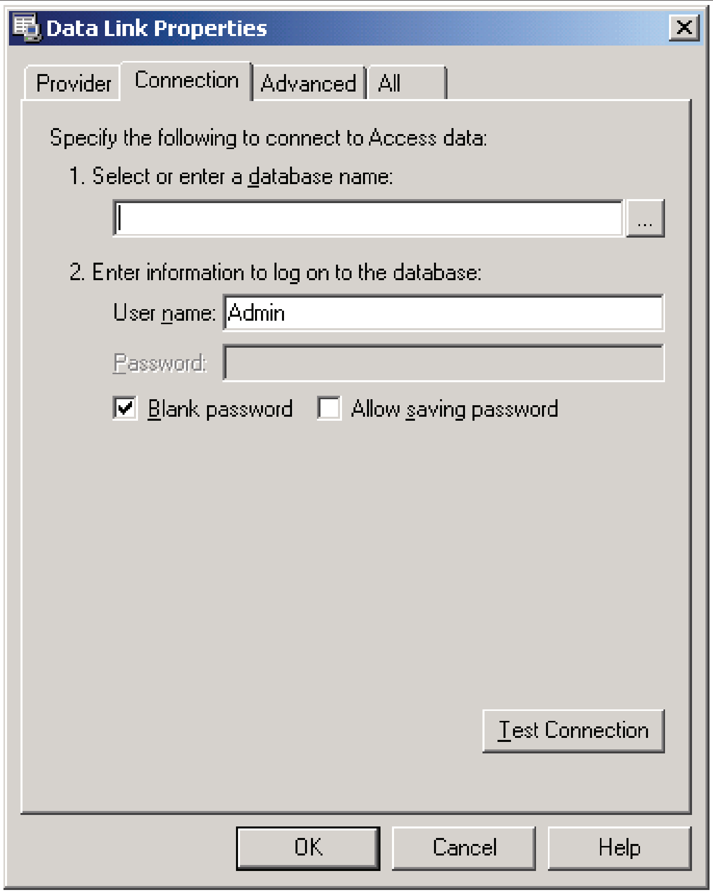
4) Right click on ADODC and select properties, the following screen will be displayed.



5) Select Use Connection String and press Build button.

6) Select driver, say Microsoft Jet OLEDB provider. Press Next button.

7) Enter or browse database name & press test connection if test connection is successful, press OK.



Specify the following to connect to Access data:

1. Select or enter a database name:

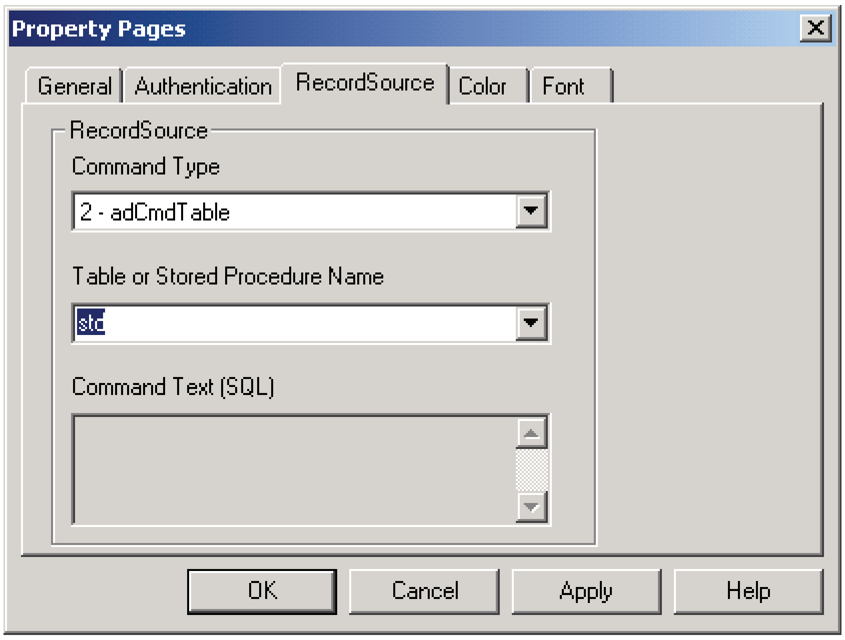
2. Enter information to log on to the database:

User name: Admin

Password:

☑ Blank password ☐ Allow saving password

8) Press Record Source tab on property window and select command type cmdTable.



RecordSource

Command Type: 2 - adCmdTable

Table or Stored Procedure Name: std

Command Text (SQL)

9) Press Apply and then OK button.

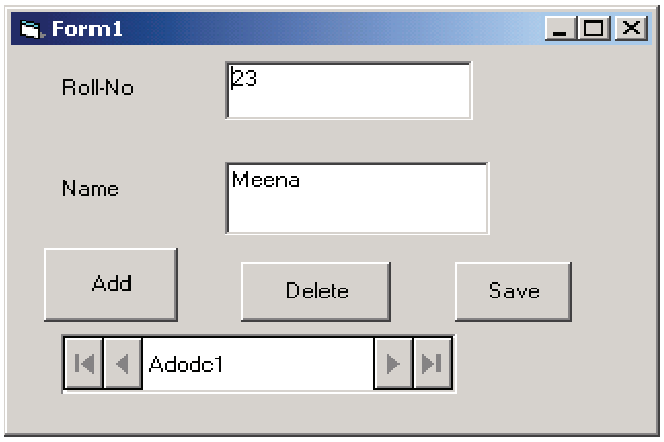
10) Drag and Drop bound control on form (say text boxes) and change following properties.

Data field	field name
Data source	ADODC name

11) Drag and Drop three buttons add, save and delete. Add following code corresponding to each button

```
Private Sub cdmdAdd_Click()
        Adodc1.Recordset.AddNew
        Text1.SetFocus
        cmdSave.Enabled = True
End Sub
Private Sub CmdDelete_Click()
        Adodc1.Recordset.Delete
        Adodc1.Recordset.MovePrevious
End Sub
Private Sub cmdSave_Click()
        Adodc1.Recordset.MoveLast
        Adodc1.Recordset(0) = Val(Text1.Text)
        Adodc1.Recordset(1) = Text2.Text
        Adodc1.Recordset.Save
        cmdAdd.Enabled = True
End Sub
```

12) Test and Run, output screen will look like.



Roll-No: 23

Name: Meena

Self Learning Exercise-2

i. Fill in the blanks

h) DAO Stands for............

i) ADO stands for ....

j) RDO stands for ....

k) RDO can easily access data stored in a variety of .... Location.

ii. True/False

h) In Forward Only Recordset, MoveFirst, MovePrevious, Find methods do not work on the Recordset.

i) RDO do not support any rdoResultset type that return a table.

j) Refresh can not be used to move to first record in a recordset.

k) Refresh can be used to open database.

5.4 Summary

This unit presented many topics, those are centered on how Visual Basic implements the relational database model through use of Jet Engine and makes the database easy to use through the data control. The chapter also introduced the data manager tool for building database tables.

First we discussed Sequential and Random Access data files. After that Database files were discussed.

The data control allows easy access to VB and other databases. Important properties are,

DataBaseName	identifies the path to the database.
RecordSource	identifies whether the data control will contain data from table or a query.
	RecordSetType identifies whether the data contained by the data control will be an updateable data set from a table or query or read-only dataset.

Bound Control has two properties to be set

DataSource	identifies which data control
DataField	identifies field in the data control Recordset.

Other methods of database connectivity were also discussed which are DAO, ADO, RDO.

DAO Jet Engine is used. In this, the methods are,

OpenRecordset	To create Recordset.
OpenDatabase	To open databases.

RDO is used to access Remote Object.

ADO is an object oriented interface which provides facility to access database using OLEDB data providers. ADO provides fast and easy access to data, We have discussed ADO data control which provides design time connectivity.

5.5 Glossary

Data	Data are known facts.
Database	Collection of data.
DBMS	Collection and management of data is known as Data Base Management System
RDBMS	Is a system in which many related database tables can be managed
Table	A table is a logical structure used to group sets of related information
Field	A field is an attribute of a record.
Primary Key	Attribute which can uniquely identify a record.
Foreign Key	Foreign keys are fields in common between tables.
Queries	Queries are questions asked for database.

5.6 Further Readings

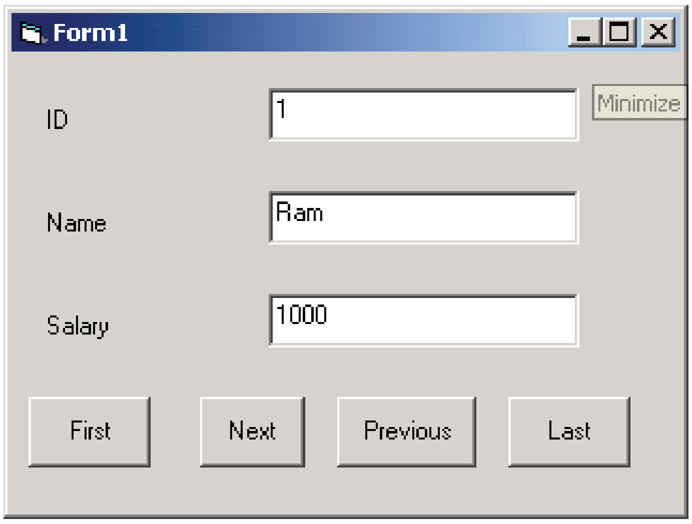
1. Alan Eliason, Ray Malarkey, “Visual Basic 6, Environment, programming and Application”.
2. Brain Siller and Jeff Spotts “Visual Basic 6”.

5.7 Answers to Self Learning Exercises

i. Fill in the blanks	ii. True/False
a) delete	a) True
b) end - of - file	b) True
c) formatted	c) False
d) raw data	d) False
e) File pointer	e) False
f) Location	f) True
g) File	g) True
h) Data Access Object	h) True
i) Active Data Object	i) True
j) Remote Data Object	j) False
k) Remote	k) True

5.10 Unit End Questions

1. Create the following interface and add Buttons corresponding to movenext, movefirst, moveprevious, movelast. Write the code To enable the navigations in the above example interface.



2. Add four buttons of caption New, Save, Edit, Delete to above interface. And write code corresponding to add new record, save entries, edit already existing record and delete a record using ADODC.

Unit - 6: Introduction to Database

Structure of the Unit

6.0 Objectives

Main objectives of this unit are to,

- Know how to create a report.
- Know how to display data in Grid form
- Know how to create MDI application
- Understand meaning of OLE and to know how to use OLE custom control
- Know how to capture and handle runtime errors.

6.1 Introduction

You have become familiar about database programming using Visual Basic. Some advanced concepts which are very useful in application development are left for discussion, for example we want an application which includes more than one document.

We want to include documents of other applications such as Microsoft Excel, Word, PaintBrush etc in our application

How we can handle some Runtime errors such as divide by zero, file does not exist etc. so that we can display appropriate messages in place of system's run time error messages which are not understandable in most of the cases.

We want to display data of database in presentable form with required format. Data Report provides this facility. How we can create report will be discuss in this chapter.

Sometimes there is requirement to display data in tabular form, since tabular form is the best way to display data, how this requirement will be fulfilled will also be discussed in current chapter.

6.2 Database Report

If you want a hardcopy of a database or selected amount of information of a table then printing of database file is not a better solution since it will not be a formatted and presentable document. Therefore some tool should be there for this requirement, Database Report is the solution.

Despite the fact that online information systems always promise a "paperless" office, everyone likes to have printed reports. In Visual Basic, Database Report enables you to easily display a print preview screen, with print and explore button, from an ADO data source. Visual Basic also supports Crystal Report a popular reporting tool that has been packaged with Visual Basic.

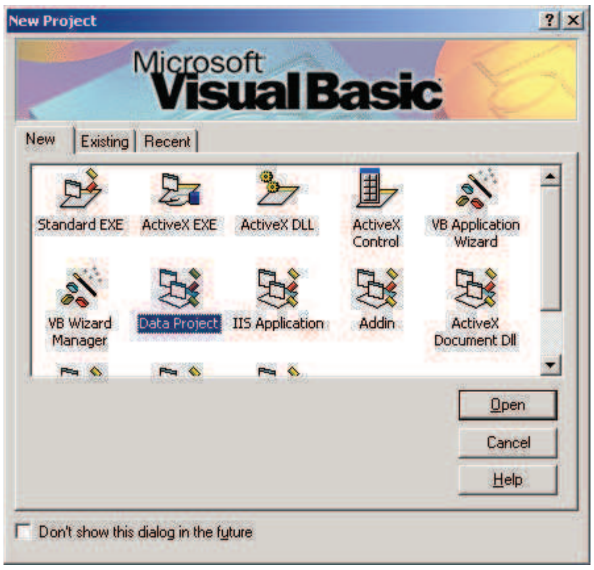
6.2.1 Sections of Database Report

Database Report is divided into several sections which are given below-

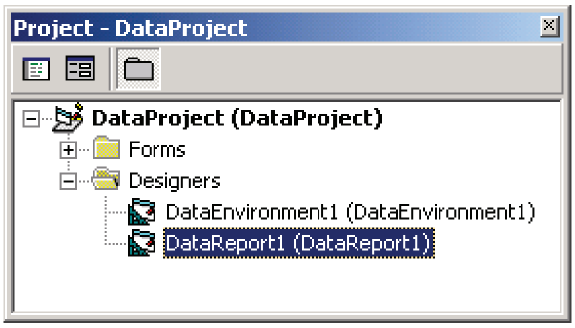
Report Header	Information displayed once, at top of the report.
Page Header	Information displayed at top of every page.
Detail Section	The part of report that is repeated for each record in the ADO data source.
Page Footer	Information displayed at the bottom of every page.
Report Footer	Information displayed once at end of the report

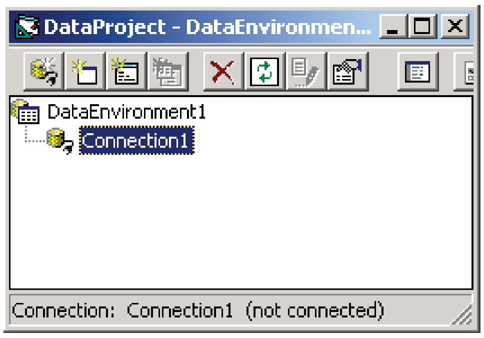
6.2.2 Steps to Create a Report from a Database

Step 1: Open a new project and select data project from the following window.

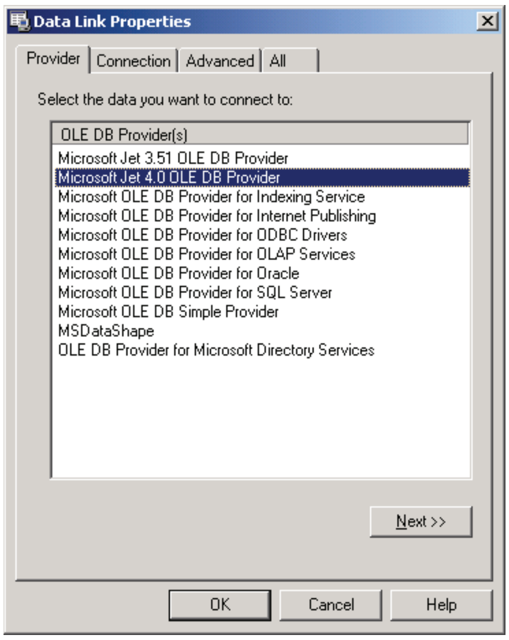


Step 2: Double click on Data Environment in project explorer window.


Project - DataProject
DataProject (DataProject)
Forms
Designers
DataEnvironment1 (DataEnvironment1)
DataReport1 (DataReport1)


DataProject - DataEnvironmen...
DataEnvironment1
Connection1
Connection: Connection1 (not connected)

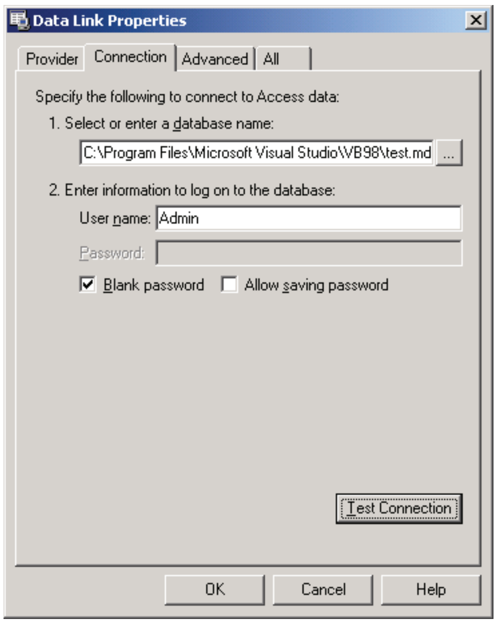
Step 3: Right click on connection, select the properties.



Select the data you want to connect to:

OLE DB Provider(s)
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services

Step 4: Select the database provider - say Microsoft Jet 4.0 OLEDB and press next.



Specify the following to connect to Access data:

1. Select or enter a database name:

C:\Program Files\Microsoft Visual Studio\VB98\test.md

2. Enter information to log on to the database:

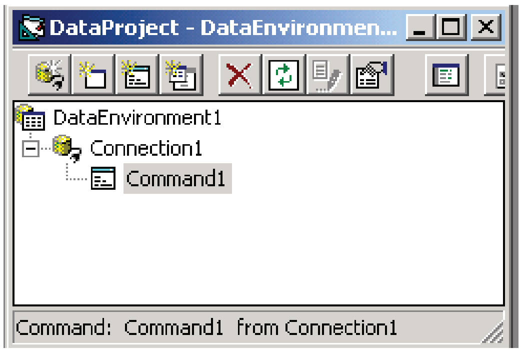
User name: Admin

Password:

[✓ Blank password] Allow saving password

Step 5: Type or browse database name, test connection and press OK.

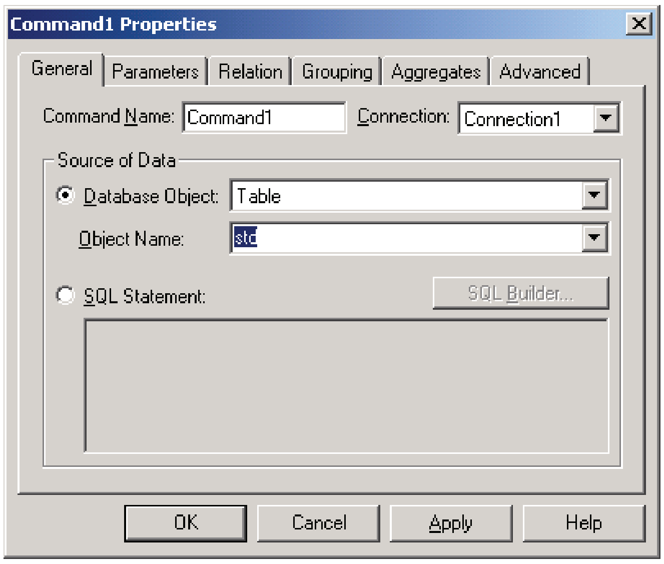
Step 6: Right click on connection, select Add command.



```
DataEnvironment1
  Connection1
    Command1
```

Command: Command1 from Connection1

Step 7: Right click Command and select Properties.



Command Name: Command1 Connection: Connection1

Source of Data

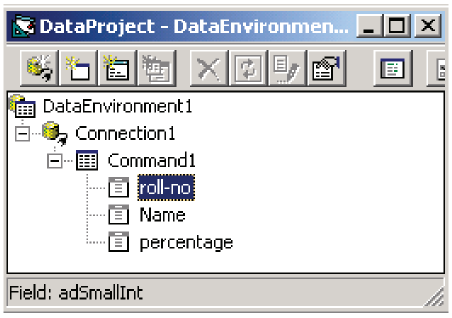
[✓ Database Object:] Table

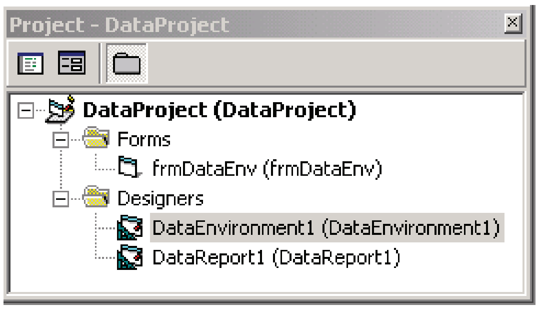
Object Name: std

SQL Statement:

Step 8: Select table from list of database objects and select table name from list of object name, and press OK.

Step 9: Expand command in Data environment window. Expand the form in project explorer window and both following screens should be displayed.



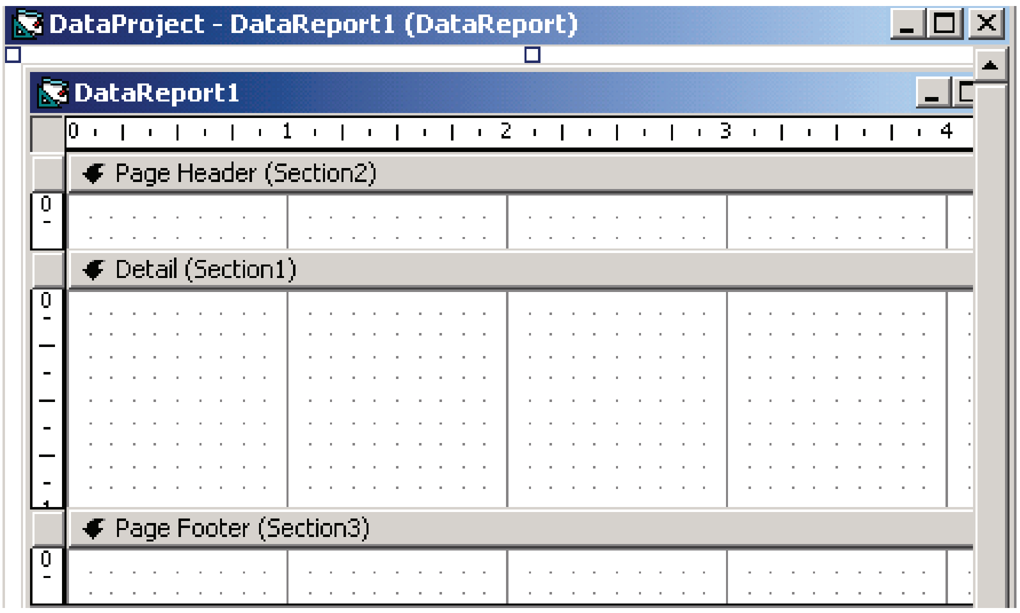

Project - DataProject
DataProject (DataProject)
Forms
frmDataEnv (frmDataEnv)
Designers
DataEnvironment1 (DataEnvironment1)
DataReport1 (DataReport1)

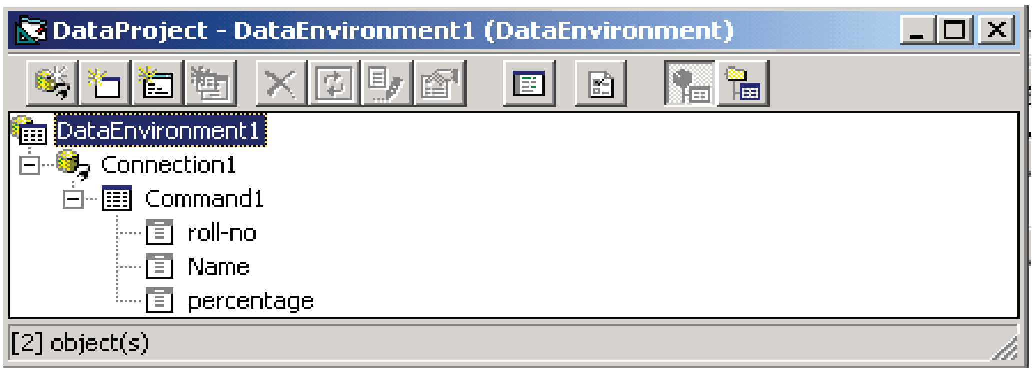
Step 10: Drag and Drop fields from data environment window to form window.

Step 11: Double click on Data Report1 in data project window. And set following data report properties,

Data Source	Data Environment1
Data member	Command1

Step 12: Now click on data Environment and Check that following windows are displayed on screen.





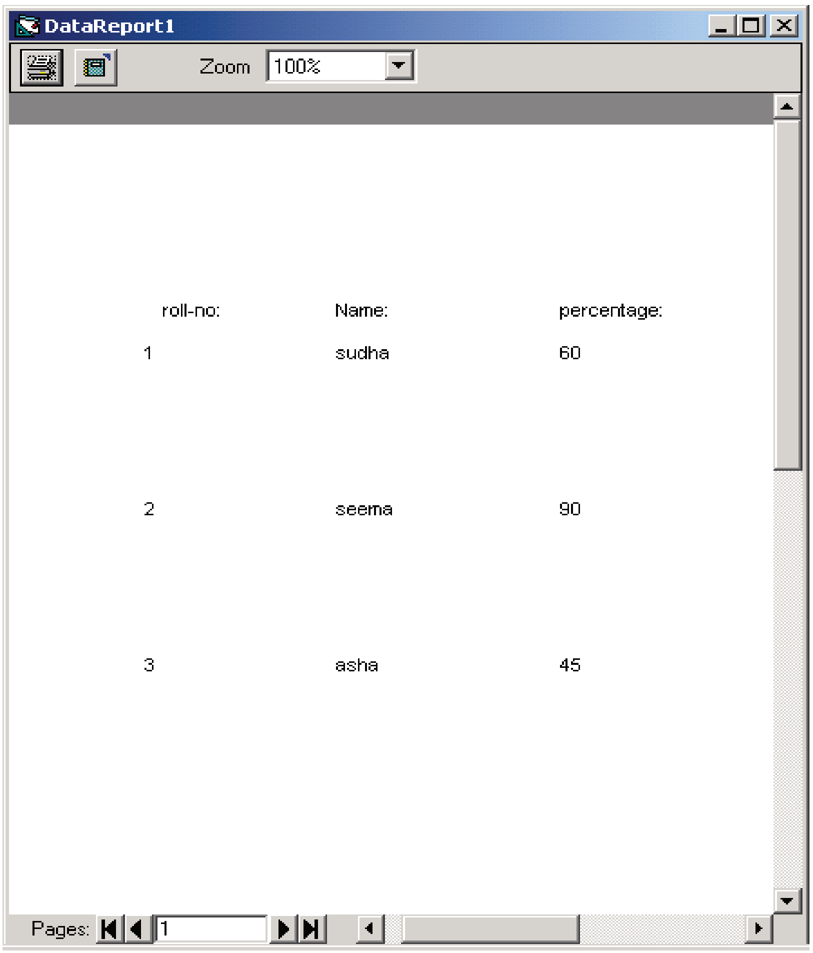
```
DataEnvironment1
  Connection1
    Command1
      roll-no
      Name
      percentage
[2] object(s)
```

Step 13: Drag and Drop fields from data environment window to Data report1.

Step 14: Add a button of Caption report to form and write following code to click event of that button.

```
Private Sub report_Click()
        DataReport1.Show
End Sub
```

Step 15: Run the project. Output screen will be displayed as follows.



```
roll-no:      Name:      percentage:
1             sudha      60
2             seema      90
3             asha       45
```

Self Learning Exercise-1

i. Fill in the blanks

a) Default name of data report is ....

b) Method used for display of data report is ....

c) .... and .... properties of data report that have to be set.

ii. True/False

a) Data Report has to combine with data environment if you want to use it.

b) We can export data report in other formats also, such as HTML

c) We can view objects of other Applications through Data Report.

6.3 Displaying Data in Grid

Information displayed in tabular form that is in rows and columns is known as Grid. Data Grid fills the rows from a data source in a table. It is used to retrieve and update the database records.

6.3.1 What is Data Grid?

Data Grid is a control which allows you to display data in the form of grid of columns and rows. Depending on how data grid is set up, the user can edit the cells of the grid directly, and change the underlying data.

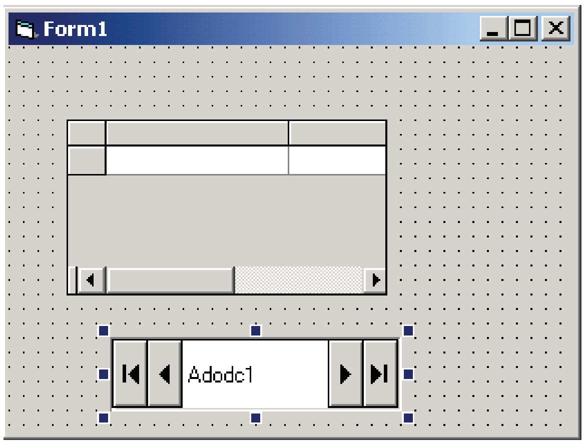
6.3.2 Creation and Data Connectivity of Data Grid

Since you will not find Data Grid control on the ToolBox therefore, to add Data Grid control on toolbox follow the steps given below,

1) Right click on empty area of ToolBox
2) Select component from context menu
3) Check Microsoft Data Grid Control 6.0 check box.
4) Press OK, close Component Dialog Box.

Data connectivity to Grid can be possible after the execution of the following steps.

Step 1: Drag and Drop Data Grid Control on the form and Add ADODC control as given on the following Screen-

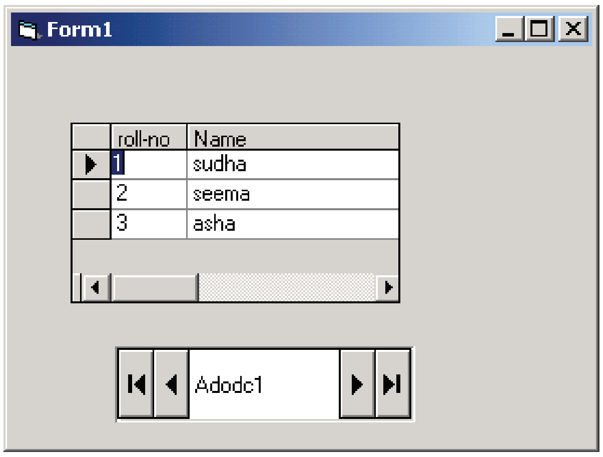


Step 2: Set properties of ADODC as described in chapter 5.

Step 3: Set the Data Source property to ADODC control name. Data Grid configures its columns automatically based on the data source.

Step 4: Right click on Grid and select Retrieve Fields. Heading row of grid will be filled by field names of records of database.

Step 5: Run the project. Output Screen will be displayed as follows,



	roll-no	Name
▶	1	sudha
	2	seema
	3	asha

Self Learning Exercise-2

i. Fill in the blanks

d) To bind the **DataGrid** control to a recordset, set its ....property and optionally, the ....property.

e) Grid Control has .... and .... properties to represent location of the currently active cells.

f) T o represent total number of rows and columns of grid properties are .... and ....

ii. True/False

d) We can Connect ADO Data Control with data Grid.

e) We can also display result of query in Data Grid.

f) We can Connect Data Control with data Grid.

6.4 Validation and Error Trapping

As you are writing code, Visual Basic informs you of syntactical errors. However once the program is running, you may encounter unexpected runtime errors in many circumstances. For example you try to open a file that a user has already deleted you will get *Run Time Error-File Not Found*.

In this situation Visual Basic stops execution of the application and displays an error message. Programmer can also write code to handle these types of errors according to his /her own way. For that he has to follow the following steps,

i) Set Error trap

ii) Write error handler

iii) Exit from error handler

i) Set Error Trap

On Error statement is used to trap an error. *On Error* statement interrupts the normal flow of your program when an error occurs and begins execution of error handler. It can be used in the following ways,

On Error GoTo (Label):	Control goes to the statement whose label is specified.
On Error Resume Next:	Control should be transferred to the statement that follows the

	error. In other words program is allowed to continue even though it contains a run-time error.
On Error GoTo 0(Zero)	This statement disables any enabled error handler.
On Error GoTo <label>	is explained by following example-

```
Private Sub disp()
        On Error Go To DivByZero
        i = n1/n2
        Print i
        Exit Sub
        DivByZero: MsgBox "Error : Divide by zero"
    End Sub
```

ii) Error handler

An error handler should provide information of error and its solution. And also it should also allow the user to execute remaining work. ErrObject will help programmer in writing error handler. Which contains information of an error and it's properties. Some properties of ErrObject are,

Number	Error's Identification number.
Description	Information about error i.e. which error it is.
Source	Object or Application name in which error occurs.

Methods associated with ErrObject are,

Raise If in a series of procedures, all procedures don't have error handler

Then, Raise method is used to send error from one procedure to another to handle it.

Syntax	Err.Raise number, Source, destination, helpfile, helpcontext
Example	Err.Raise number = 101

Clear	To set error number to zero.

Example

```
Private Sub Form-Load()
            On Error GoTo err1
            ........
            ......
            Exit Sub
            err1:
                MsgBox err.number
                MsgBox err.description
                MsgBox err.Source
        End Sub
```

iii) Exiting an error handler

Resume or Resume Next is used to exit from error handler. Exit Sub statement can also be used to prevent execution of error handler.

Self Learning Exercise-3

i. Fill in the blanks

g) Errors arise during running of program is known as ……… errors.

h) Default property of Err object is …………..

i) ……… statement allows to stop runtime error.

ii. True/False

g) Default property of Err object is description.

h) File not found is an example of Run time Error.

i) On Error statement is used to trap an error.

6.5 Multiple Document Interface

Multiple Document Interface (MDI) allows programs to work with multiple forms contained within a parent form. The MDI enhances programs into two ways. First you can have one container form that acts as the background for your overall application. If a user moves the container form the child form contained inside moves as well, which helps keep your application interface organized and self contained. Second perhaps even more powerful, your user can work on multiple documents at one time. MDI application allows the use of multiple interfaces of the same form, which can add a great deal of power and flexibility to your program.

Steps to Create MDI Application

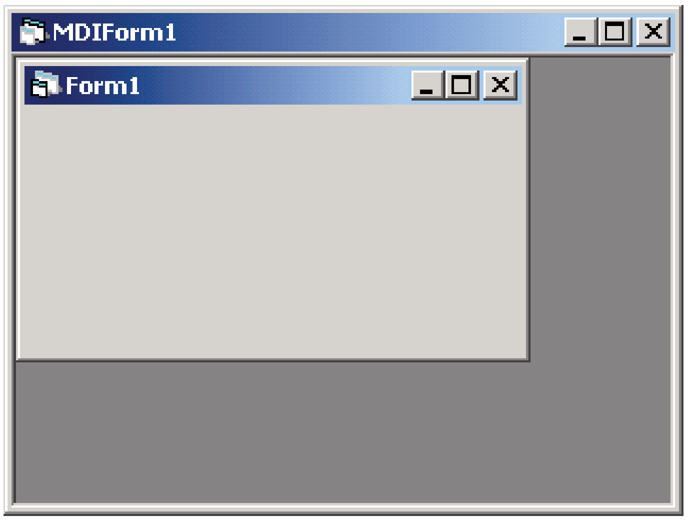
In this section, you will walk through the steps of creating a simple MDI application using Visual Basic. You can create an MDI application using the following steps,

Step 1: Create a new project

Step 2: Create a MDI parent form by clicking Add MDI form option in project menu item.

Step 3: Create a standard form and set its MDI child property to True.

Step 4: Run the program to see how child forms behave inside the parent. Child forms will be displayed only within the area of parent form as displayed on following screen,



AutoShowChildren and ScrollBars are special properties of MDI form. AutoShowChildren determines whether child forms are shown automatically as they are loaded that means their Load statement and Show statement have same effect on form.

The ScrollBars property determine whether MDI form shows scrollbars when necessary. When this property is True, Scrollbars are shown on MDI form if one or more child form extends beyond the boundary of MDI parent form.

Self Learning Exercise-4

i. Fill in the blanks

j) The SDI stands for …………

k) The MDI stands for ……………

l) A VB program can have …………. MDI forms.

m) Forms that are contained within MDI form are referred as…………..

ii. True/False

j) If AutoShowChildren property of MDI Form is True that means Load statement and Show statement have same effect on form.

k) To make a form child form, we set its MDI child property to False.

l) MDI allows programs to work with multiple forms contained within a parent form.

m) If a user moves the container form the child form contained inside does not.

6.6 Object Linking and Embedding (OLE)

OLE is the method for displaying and manipulating data found in other window applications for example, Microsoft Word, Excel etc. With Object linking, VB application is linked by path name to another windows application. However, other window application called server application, stores and maintains linked data.

With Object Embedding Source data from server application is actually stored in VB application called the client application.

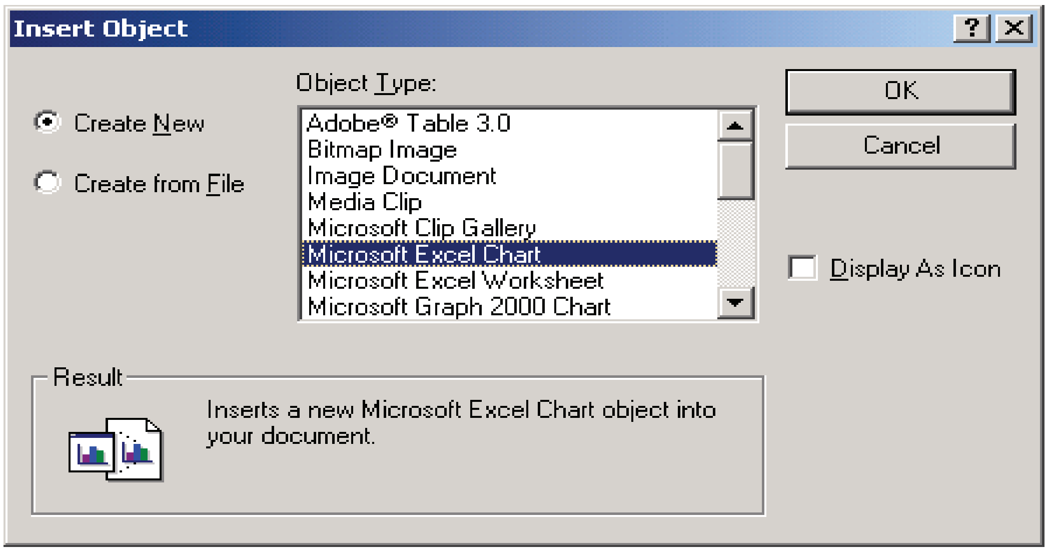
The simplest and quickest way to use OLE is by making use of the OLE container control, provided as standard by Visual Basic. With no programming, you can draw one of these controls onto your Visual Basic form, set the Source to point to the file of your choice and you have created an OLE client application.

6.6.1 Object Linking at Design Time

Step 1: Start new project.

Step 2: Double click OLE Control.

Step 3: An Insert Dialog Box will appear as shown below,



Adobe® Table 3.0
Bitmap Image
Image Document
Media Clip
Microsoft Clip Gallery
Microsoft Excel Chart
Microsoft Excel Worksheet
Microsoft Graph 2000 Chart

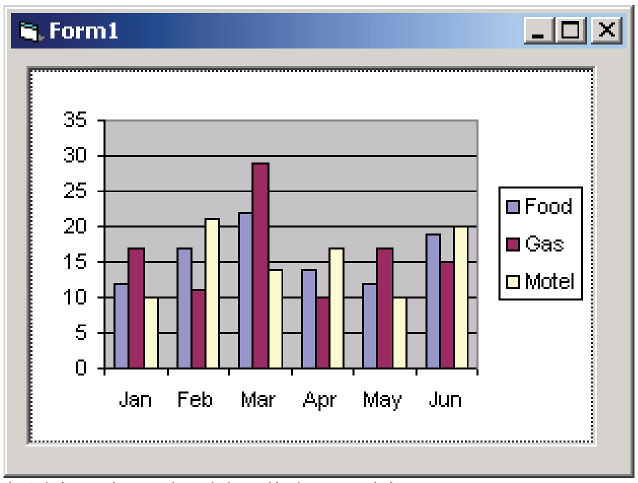
Inserts a new Microsoft Excel Chart object into your document.

Step 4: Select the source application of which object you want to add

OR

Click create from file type folder in which required file is stored.

Step 5: Object will be added to form Run the application out put will be as shown in following figure,



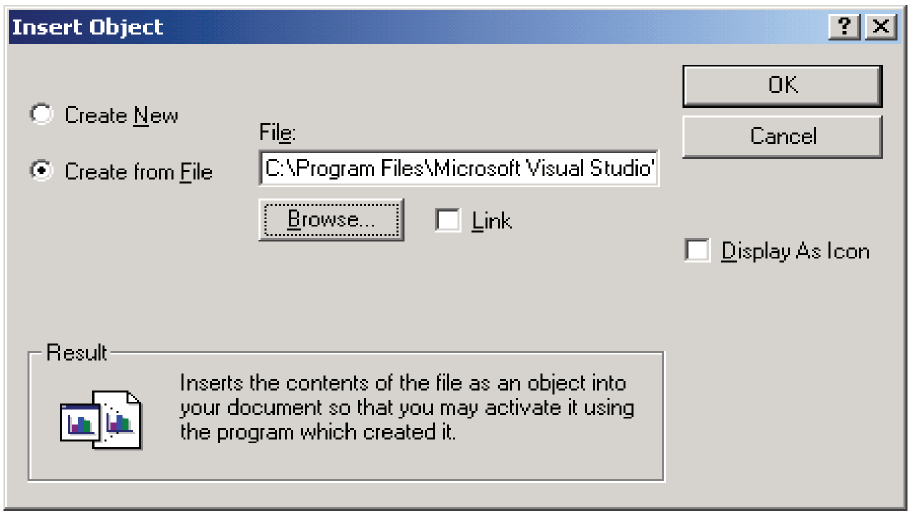
Step 6: To edit added Object just double click on object.

6.6.2 Object Embedding at Design Time

Step 1: Create a new Project

Step 2: Add OLE Control to FORM

Step 3: Click create from File. The following window will appear on screen,



File: C:\Program Files\Microsoft Visual Studio'

Result: Inserts the contents of the file as an object into your document so that you may activate it using the program which created it.

Step 4: Set the path where file is located but Do not click Link.

Step 5: Set Sizemode of OLE1 control to Autosize.

Step 6: Test and Run The application.

Self Learning Exercise-5

i. Fill in the blanks

n) OLE is associated with three concepts, Object, server and ……….

o) OLE stands for ………………

ii. True/False

n) In Object Linking client stores the data.

o) In Object Embedding means server stores the data.

6.7 Summary

Data Report enables you to easily display a print preview screen with print button with no programming or less programming.

Information presented in the form of tables (rows and columns) is known as Grid. Data Grid and MSFlexGrid are controls which allow programmer to display information in Grid form. We can also bind grid to a database table to display complete table of records in grid form with no programming.

Error handling is added to file processing code to trap errors such as, drive not responding to a request, or a file does not exist. VB contains a large number of trappable errors. The instruction OnError GoTo ErrorSection must be matched with label, with ErrorSection found later in the code

MDI allows user to work on more than one document at one time.

Object Linking means to link the client application to the server application, with the server storing the data.

Object Embedding means to link the client application to server application with the client storing the data. With object embedding the link path is set; however, the source data is bound to the VB application.

6.8 Glossary

Data Grid	is a control which allows you to display data in the form of grid of columns and rows.
MDI Application	allows programs to work with multiple forms contained within a parent form.
OLE	is the method for displaying and manipulating data found in other windows applications.

6.9 Further Readings

1. Steven Holzner, "Visual Basic 6 Programming Black Book", Dreamtech Press, New Delhi.
2. Evangelas Petroutsos,"Mastering Visual Basic 6", BPB Publications, New Delhi
3. Deitel, Deitel, "Visual Basic 6 How to program", Pearson Education
4. Noel Jerke," The Complete Reference VB 6", TataMcgraw Hill

6.10 Answers to Self Learning Exercises

i. Fill in the Blanks	ii. True/ False
a) Data report1	a) True
b) Show	b) True
c) Data Source and Data member	c) True
d) Data Source, Data member	d) True
e) Row, Col	e) True
f) Rows, Cols	f) False
g) Runtime	g) False
h) Number	h) True
i) On Error	i) True
j) Single Document Interface	j) True
k) Multiple Document Interface	k) False
l) Single	l) True
m) MDI Child Form	m) False
n) Client	n) False
o) Object Linking and Embedding	o) False

6.11 Unit End Questions

1. Create a MDI application and add a menu item File having new & exit options. Create instance of an editor corresponding to new button and close all open document in exit option.
2. Trap and handle error of file does not exit if you are trying to read a file which has either been deleted or does not exists.
3. Create the following interface. Write a code to add new records to grid corresponding to add button; and clear text boxes to allow the next student's record to be added to the grid.

Unit - 7: Graphics in Visual Basic

Structure of the Unit

7.0 Objectives

After reading this chapter you will be able to understand,

- The basic concepts of Graphics and animation in VB
- ActiveX concepts and implementation
- Concepts of Dynamic Link Libraries
- Multimedia in VB6.0
- MFC library and
- Object Oriented Features of Visual Basic

7.1 Introduction

This chapter is on one of the most popular topics in Visual Basic—graphics. A branch of Computer Science that deals with the theory and techniques of computer image synthesis. Computer graphics is used to enhance the transfer and understanding of information in science, engineering, medicine, education, and business by facilitating the generation, production, and display of synthetic images of natural objects with realism almost indistinguishable from photographs. Computer graphics facilitate the production of images that range in complexity from simple line drawings to three-dimensional reconstructions of data obtained from computerized axial tomography (CAT) scans in medical applications. Graphics has became essential part of computer science as well as human beings who interact with computers. Here, we'll cover how to use graphics in Visual Basic. There's a great deal of graphics power in Visual Basic.

7.2 Dragging and Dropping Multiple Objects

Windows users fall into two general categories: those who prefer to use the keyboard and those who prefer to use the mouse. Programmers have been taught to look after the needs of keyboard users by providing access keys (the underlined letter in a command or menu) and shortcuts (such as a CTRL + letter combination), but the needs of mouse users have largely been ignored. Programmers tend to primarily be keyboard users, so the emphasis on keyboard-oriented features is understandable, but every programmer should consider providing mouse support as well.

One thing that mouse users expect is the ability to drag and drop. If you look at most major applications or at Windows itself, drag and drop is everywhere. For example, users are accustomed to dragging and dropping files in the Windows Explorer and to dragging and dropping text in Microsoft Word.

In graphical user interfaces, drag-and-drop is the action of (or support for the action of) clicking on a virtual object and dragging it to a different location or onto another virtual object. In general, it can be used to invoke many kinds of actions, or create various types of associations between two abstract objects.

Dragging requires more physical effort than moving the same pointing device without holding down any buttons. Because of this, a user cannot move as quickly and precisely while dragging. However, drag-and-drop operations have the advantage of thoughtfully chunking together two operands (the object to drag, and the drop location) into a single action. Extended dragging and dropping (as in graphic design) can stress the mousing hand. As a feature, support for drag-and-drop is not found in all software, though it is sometimes a fast and easy-to-learn technique for users to perform tasks.

A design problem appears when the same button selects and drags items. Imprecise movement can cause a dragging when the user just wants to select. Another problem is that the target of the dropping can be hidden under other objects. The user would have to stop the dragging, make both the source and the target visible and start again.

How Drag and Drop Works?

Drag and drop is actually the same as cutting and pasting (or copying and pasting) using the mouse instead of the keyboard. In both cases you have a source (where you are cutting or copying from) and a target (where you are pasting to). During either operation, a copy of the data is maintained in memory. Cut and paste uses the Clipboard; drag and drop uses a DataObject object, which is in essence a private clipboard.

Self Learning Exercise-1

i. Fill in the Blanks

a) Dragging is initiated by calling the ________ method.

ii. State True/False

a) Drag and drop is actually the same as cutting and pasting (or copying and pasting) using the mouse instead of the keyboard.

7.3 Graphics

There are two principal ways of drawing graphics in Visual Basic: using graphics methods and using graphics controls (like the line and shape controls). Graphics methods work well in situations where using graphical controls requires too much work. For example, creating gridlines on a graph would require an array of line controls but only a small amount of code using the Line method. In addition, when you want an effect to appear temporarily, you can write a couple of lines of code for this temporary effect instead of using another control. Also, graphics methods offer some visual effects that are not available in the graphical controls. For example, you can only create arcs or paint individual pixels using the graphics methods.

Visual Basic Coordinate System for Graphics

We can draw figures in forms and controls like picture boxes, so we should know how measurements and coordinates are set up in those objects. Visual Basic coordinate systems have the origin (0, 0) at upper left and are specified as (x, y), where x is horizontal and y is vertical (note that y is positive in the downwards direction). When we draw graphics in Visual Basic, we'll be using this coordinate system. The default unit of measurement in Visual Basic is twips (or 1/1440s of an inch). That unit was originally chosen to be small enough to be device-independent, but if you don't like working with twips, you can change to other measurement units like millimeters, inches, and so on.

7.4 Layering Graphics

When you create graphics in Visual Basic, bear in mind that graphical controls and labels, nongraphical controls, and graphics methods appear on different layers. The behavior of these layers depends on three things: the AutoRedraw property, the ClipControls property, and whether graphics methods appear inside or outside the Paint event. Usually the layers of a form or other container are as follows:

- **Front layer:** Nongraphical controls like command buttons, checkboxes, and file controls.

- **Middle layer:** Graphical controls and labels.
- **Back layer:** Drawing space for the form or container. This is where the results of graphics methods appear.

Anything in one layer covers anything in the layer behind it, so graphics you create with the graphical controls appear behind the other controls on the form, and all graphics you create with the graphics methods appear below all graphical and nongraphical controls. Combining settings for AutoRedraw and ClipControls and placing graphics methods inside or outside the Paint event affects layering and the performance of the application. You can find the effects created by different combinations.

Table 7.1: Combination of AutoRedraw, ClipControls and Paint

AutoRedraw	**ClipControls**	**Method In/Out Paint Event**	**Description**
True	True(Default)	**Paint** event Ignored	Normal Layering
True	False	**Paint** event Ignored	Normal Layering Forms with many controls that do not overlap may paint faster because no clipping region is calculated or created.
False (Default)	True (Default)	In	Normal Layering
False	True	Out	Nongraphical control in front. Graphical Control and Graphical method appear mixed in the middle and back layer.
False	False	In	Normal Layering, affecting only pixel that were previously covered.
False	False	Out	Graphics Methods and all controls appear mixed in the three layers.

7.5 Simple Animation

The simplest way you create animation is by displaying successive frames of the animation sequence at intervals. We use timer control for this purpose. That is a good job for the timer control. To see how this works, we will create an example now. In this example, we will just switch back and forth between two simple images, image1.bmp and image2.bmp, which are simply strips of solid color, red and blue.

To store those images in our program, add an image list control, ImageList1. You add image list controls with the Project|Components menu item; click the Controls tab in the Components dialog box that opens, select the Microsoft Windows Common Controls item, and click on OK to close the Components box. Draw a new image list control, ImageList1, and right-click it, selecting Properties in the menu that opens. We click the Images tab in the image lists property pages, and we use the Insert Picture button to insert the two images in image1.bmp and image2.bmp into the image list.

Next, add a timer control, Timer1; set its Interval property to 1000 (in other words, 1 second), and set its Enabled property to False. Also add a command button, Command1, with the caption

'Start Animation', and a picture box, Picture1, setting the picture box's AutoSize property to True so that it resizes itself to fit our images. That's it, now we are ready to write some code. We start the animation when the user clicks the 'Start Animation' button by enabling the timer:

```
Private Sub Command1_Click()
Timer1.Enabled = True
End Sub
```

We will keep track of the current image with a Boolean variable named blnImage1; if this Boolean variable is True, we will display the first image in the image list:

```
Private Sub Timer1_Timer()
Static blnImage1 As Boolean
If blnImage1 Then
Picture1.Picture=ImageList1.ListImages(1).Picture
...
```

Otherwise, we will display the second image in the image list:

```
Private Sub Timer1_Timer()
Static blnImage1 As Boolean
If blnImage1 Then
Picture1.Picture=ImageList1.ListImages(1).Picture
Else
Picture1.Picture=ImageList1.ListImages(2).Picture
End If
...
```

Finally, we toggle blnImage1 :

```
Private Sub Timer1_Timer()
Static blnImage1 As Boolean
If blnImage1 Then
Picture1.Picture=ImageList1.ListImages(1).Picture
Else
Picture1.Picture=ImageList1.ListImages(2).Picture
End If
blnImage1 = Not blnImage1
End Sub
```

And that is s all we need. When you run the program and click the Start Animation button, the animation starts: the picture box flashes red and blue images once a second.

7.6 ActiveX

ActiveX controls and ActiveX documents are two of the ActiveX components you can build with Visual Basic. In fact, the ActiveX part of Visual Basic has exploded in scope lately, along with many changes in terminology, and will surely do so again.

The whole ActiveX field started originally to differentiate controls designed for Internet usage from general OLE (Object Linking and Embedding) controls. In time, however, all OLE controls have come to be referred to as ActiveX controls. In fact, the field has taken off so vigorously that now Visual Basic can build not just ActiveX controls in Visual Basic (you used to have to build ActiveX controls for use in Visual Basic in other programming packages, like Visual C++), but ActiveX components. What is an ActiveX component? In programming terms, all ActiveX components are really OLE servers, but that doesn't help us understand what's going on. It's better to break things down and look.

7.6.1 Types of ActiveX Components

The three types of ActiveX components:

- ActiveX Controls
- ActiveX Documents
- Code Components (OLE automation servers)

ActiveX Controls

ActiveX controls, those are the controls you can add to the Visual Basic toolbox using the Components dialog box. You can add those controls to a Visual Basic program like any other control. You can also use ActiveX controls on the Internet, embedding them in your Web pages. ActiveX controls can support properties, methods, and events.

Your ActiveX control can be built entirely from scratch (in other words, you're responsible for its appearance), it can be built on another control (such as a list box), or it can contain multiple existing controls (these ActiveX controls are said to contain constituent controls). Visual Basic ActiveX controls are based on the Visual Basic UserControl object. When you create an ActiveX control, you create a control class file with the extension .ctl. Visual Basic uses that file to create the actual control, which has the extension .ocx. After you register that control with Windows (you can use Windows utilities like regsvr32.exe to register a control), the control will appear in the Visual Basic Components dialog box, ready for you to add to a program. You can also use these controls in web pages.

ActiveX Documents

ActiveX documents are new to many programmers, but the idea is simple. Instead of restricting yourself to a single control in a web page, now you can create the whole page. With ActiveX documents, the result is just like running a Visual Basic program in your Web browser or other application.

Visual Basic ActiveX documents are based on the Visual Basic UserDocument object. When you create an ActiveX document, you save it with the extension .dob. Visual Basic uses that .DOB file to create the EXE or DLL file that holds the actual code for the ActiveX document. In addition, Visual Basic produces a specification file, with the extension .vbd, that describes the ActiveX document, and it's that file that you actually open in the host application, such as the Microsoft Internet Explorer. With ActiveX documents, you can let users save data (using the PropertyBag property); that data is stored in the VBD file.

Code Components (OLE Automation Servers)

Code components were formerly called OLE automation servers. These objects let you use their code in other programs. For example, you might have a calculation routine that you expose in a code component; doing so makes that routine available to other programs. Code components can support properties and methods.

7.6.2 In-Process V/s Out-of-Process Component

If an ActiveX component has been implemented as part of an executable file (EXE file), it is an out-of-process server and runs in its own process. If it has been implemented as a dynamic link library (DLL file), it is an in-process server and runs in the same process as the client application.

If your ActiveX component is an out-of-process server, it is an EXE file, and can run standalone. Applications that use in-process servers usually run faster than those that use out-of-process servers because the application doesn't have to cross process boundaries to use an object's properties, methods and events. There are a few reasons why you may want to create your ActiveX document as an in-process component (DLL file). The performance of an in-process component surpasses that of the same component compiled as an EXE. In addition, multiple programs accessing the same EXE can overwrite global data; that doesn't happen if they each have their own in-process server.

7.6.3 Which ActiveX Component You Want To Build?

With all the different types of ActiveX components to choose from, how do you decide which type of component you want to create? Take a look at this list:

- To build an invisible component that provides routines in code that you can call, build a code component (ActiveX EXE or an ActiveX DLL).
- To build a component that can run in the same process with your application, build an ActiveX DLL.
- To build a component that can serve multiple applications and can run on a remote computer, build an ActiveX EXE.
- To build a visible component that can be dropped into an application at design time, build an ActiveX control.
- To build a visible component that can take over an application window at runtime, build an ActiveX document.

7.6.4 Design an ActiveX Control

You can design the appearance of your ActiveX control entirely from scratch, creating an entirely new control, never seen before. In that case, you're responsible for creating the control's appearance from scratch. Later, you can add events to your control, as well as methods and properties.

To design the appearance of your entirely new control, you can use the Visual Basic graphics methods that the UserControl object supports, such as Circle, Line, PSet, Print, Cls, and Point. You can also display an image in the UserControl object by setting its Picture property.

Let's see an example. Here, we'll just draw two lines to crisscross an ActiveX control and draw a black box in the middle. Create a new ActiveX control now, and double-click it at design time to open the code window to the UserControl_Initialize function:

```
Private Sub UserControl_Initialize()

…….

End Sub
```

This function is just like the Form Load procedure that we're familiar with. Set the control's AutoRedraw property to True so we can draw graphics from UserControl_Initialize, and then draw the lines to crisscross the control, using the Line method and ScaleWidth and ScaleHeight just as you would in a Visual Basic form:

```
Private Sub UserControl_Initialize()
Line (0, 0)-(ScaleWidth, ScaleHeight)
Line (0, ScaleHeight)-(ScaleWidth, 0)
...
End Sub
```

Next, we draw a filled-in black box in the center of the control this way:

```
Private Sub UserControl_Initialize()
Line (0, 0)-(ScaleWidth, ScaleHeight)
Line (0, ScaleHeight)-(ScaleWidth, 0)
Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, _
3 * ScaleHeight / 4), , BF
End Sub
```

Let's test this new ActiveX control now in the Microsoft Internet Explorer (assuming you have that browser installed). To do that, just select the Run menu's Start item now. Doing so opens the Project Properties dialog box.

Leave UserControl1 in the Start Component box, and make sure the Use Existing Browser box is clicked, then click on OK. This registers our control with Windows, creates a temporary HTML page with the control embedded in it, and starts the Internet Explorer. Now we've created our first ActiveX control and designed its appearance from scratch. If we wanted to, we could add events, properties, and methods to this control.

Here's the temporary HTML page that Visual Basic creates to display our ActiveX control; note that our control is registered with Windows and has its own ID, so this page can use the HTML <OBJECT> tag to embed one of our controls in the page:

```
 <HTML>
<BODY>
<OBJECT classid="clsid:B2A69D3B-D38C-11D1-8881-E45E08C10000">
</OBJECT>
</BODY>
</HTML>
```

Note, however, that when you select End in the Run menu, Visual Basic unregisters our control with Windows. To register it permanently, use a Windows utility like regsvr32.exe.

7.6.5 Registering An ActiveX Control

To install an ActiveX control in Windows, you must register it with Windows, and you can do that either with the setup program or with the Window regsvr32.exe utility. Let's see an example. Here, we'll see how to register an ActiveX control named, say, activex.ocx with Windows. First, use the File menu's Make activex.ocx menu item to create activex.ocx. Next, we'll use regsvr32.exe, which is usually found in the

C:\windows\system directory,

To register that control with Windows.

Here's how to register our ActiveX control:

C:\windows\system>regsvr32 c:\vbbb\activex.ocx

After the ActiveX control is registered, it will appear in the Visual Basic Components dialog box (which you open with the Project menu's Components item), and you can add it to the Visual Basic toolbox.

7.6.6 Using A Custom ActiveX Control In A Visual Basic Program

Now that we've registered our Visual Basic control with Windows (see the previous topic), how do you add it to a form in a Visual Basic project? You add ActiveX controls that you build to Visual Basic projects just as you add any other ActiveX controls, such as the ones that come with Visual Basic.

Let's see an example. Suppose you have created a calculator ActiveX control. After we register the calculator ActiveX control, that control will appear in the Visual Basic Components dialog box. Start a new standard EXE project now and open the Visual Basic Components dialog box by selecting the Project|Components item. Next, click the Controls tab in the Components dialog box.

Click the entry labeled calculator ("calculator" was the name we gave to the project when we created this control) in the Components dialog box to add our calculator, and close that dialog box to add the calculator to the Visual Basic toolbox . Now draw a new calculator ActiveX control in the program's main form, creating the new control, CalculatorControl1. (CalculatorControl was the name we gave to the UserControl object when we created this control), and run the program.

That's it, Now we've created, registered, and added a functioning ActiveX control to a Visual Basic program.

7.6.7 ActiveX Document DLL's V/s EXE's

You understand both ActiveX document EXEs and DLLs. Here's the difference: if an ActiveX document is written as an executable file (EXE file), it is an *out-of-process* server and runs in its own process; if it has been implemented as a dynamic link library (DLL file), it is an *in-process* server and runs in the same process as the client application.

Although ActiveX documents are usually built as EXE projects, the benefit of DLLs is that applications that use in-process servers usually run faster than those that use out-of-process servers because the application doesn't have to cross process boundaries to use an object's properties, methods, and events. In addition, the performance of an in-process component, or DLL file, surpasses that of the same component compiled as an EXE. Also, multiple programs accessing the same EXE can overwrite global data, but that doesn't happen if they each have their own in-process server.

7.7 Dynamic Link Libraries (DLLs)

A dynamic link library (DLL) is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer. The small program that lets the larger program communicate with a specific device such as a printer or scanner is often packaged as a DLL program (usually referred to as a DLL file). DLL files that support specific device operations are known as device drivers. The advantage of DLL files is that, because they don't get loaded into random access memory (RAM) together with the main program, space is saved in RAM. When and if a DLL file is needed, then it is loaded and run. For example, as long as a user of Microsoft Word is editing a document, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, then the Word application causes the printer DLL file to be loaded and run.

A DLL file is often given a ".dll" file name suffix. DLL files are dynamically linked with the program that uses them during program execution rather than being compiled with the main program. The set of such files (or the DLL) is somewhat comparable to the library routines provided with programming languages such as C and C++.

The original purpose for DLLs was saving both disk space and memory required for applications by storing it locally on the hard drive. In a conventional non-shared library, sections of code are simply added to the calling program; if two programs use the same routine, the code has to be included in both. Instead, code which multiple applications share can be separated into a DLL which only exists as a single, separate file, loaded only once into memory during usage. Extensive use of DLLs allowed early versions of Windows to work under tight memory conditions.

DLLs provide the standard benefits of shared libraries, such as modularity. Modularity allows changes to be made to code and data in a single self-contained DLL shared by several applications without any change to the applications themselves. This basic form of modularity allows for relatively compact patches and service packs for large applications, such as Microsoft Office, Microsoft Visual Studio, and even Microsoft Windows itself.

Another benefit of the modularity is the use of generic interfaces for plug-ins. A single interface may be developed which allows old as well as new modules to be integrated seamlessly at run-time into pre-existing applications, without any modification to the application itself. This concept of dynamic extensibility is taken to the extreme with ActiveX.

In Visual Basic (VB), only run-time linking is supported; but in addition to using LoadLibrary and GetProcAddress API functions, declarations of imported functions are allowed. When importing DLL functions through declarations, VB will generate a run-time error if the DLL file cannot be found. The developer can catch the error and handle it appropriately.

In short DLL has two functions:

1. It permits the sharing of code. The same DLL can be used by many other DLLs and applications. The Win32 API, for instance, is implemented as a series of Windows DLLs. Moreover, as long as multiple processes load the same DLL at the same base address, they can share the code in the DLL. In other words, a single DLL in memory can be accessed by multiple processes.
2. It allows for component-based and modular development, which makes development and the upgrade process easier.

7.7.1 Using DLL Procedure in Your Application

Because DLL procedures reside in files that are external to your Visual Basic application, you must specify where the procedures are located and identify the arguments with which they should be called. You provide this information with the Declare statement. Once you have declared a DLL procedure, you can use it in your code just like a native Visual Basic procedure.

When you call any DLLs directly from Visual Basic, you lose the built-in safety features of the Visual Basic environment. This means that you increase the risk of system failure while testing or debugging your code. To minimize the risk, you need to pay close attention to how you declare

DLL procedures, pass arguments, and specify types. In all cases, save your work frequently. Calling DLLs offers you exceptional power, but it can be less forgiving than other types of programming tasks.

In the following example, we'll show how to call a procedure from the Windows API. The function we'll call, SetWindowText, changes the caption on a form. While in practice, you would always change a caption by using Visual Basic's Caption property, this example offers a simple model of declaring and calling a procedure.

Declaring a DLL Procedure

The first step is to declare the procedure in the Declarations section of a module:

```
Private Declare Function SetWindowText Lib "user32" _
Alias "SetWindowTextA" (ByVal hwnd As Long, _
ByVal lpString As String) As Long
```

You can find the exact syntax for a procedure by using the API Viewer application, or by searching the Win32api.txt file. If you place the Declare in a Form or Class module, you must precede it with the Private keyword. You declare a DLL procedure only once per project; you can then call it any number of times.

Calling a DLL Procedure

After the function is declared, you call it just as you would a standard Visual Basic function. Here, the procedure has been attached to the Form Load event:

```
Private Sub Form_Load()
SetWindowText Form1.hWnd, "Welcome to VB"
End Sub
```

When this code is run, the function first uses the hWnd property to identify the window where you want to change the caption (Form1.hWnd), then changes the text of that caption to "Welcome to VB."

Remember that Visual Basic can't verify that you are passing correct values to a DLL procedure. If you pass incorrect values, the procedure may fail, which may cause your Visual Basic application to stop. You'll then have to reload and restart your application. Take care when experimenting with DLL procedures and save your work often.

7.7.2 Types of Windows DLL

- Advapi32.dll: Advanced API Services library supporting numerous APIs including many security and Registry calls
- Comdlg32.dll: Common Dialog API library
- Gdi32.dll: Graphics Device Interface API library
- Kernel32.dll: Core Windows 32-bit base API support
- Lz32.dll: 32-bit compression routines
- Mpr.dll: Multiple Provider Router library
- Netapi32.dll: 32-bit Network API library
- Shell32.dll: 32-bit Shell API library
- User32.dll: Library for user interface routines

- Version.dll: Version library
- Winmm.dll: Windows Multimedia library

7.7.3 Advantages of Creating and Using DLL's in VB6.0

A Dynamic Link Library (DDL) is a file that contains compiled code. In this sense, it's like an executable program (i.e., an EXE file), but there is one major difference: Once a DLL is installed and registered on a system, its code is available to any program that calls it. This can be an advantage when you're writing multiple programs that will usually be installed together. By placing shared code in a DLL, you can decrease program size and simplify the task of upgrades, and also make the code easily reusable in other projects.

Another use for DLLs is in Web programming. Active Server Page (ASP) technology lets you include code in your Web pages to provide dynamic functionality and database access, but the ASP code is viewable by anyone who visits your site. If you want to hide your code so that others cannot "borrow" it for their own use, you can create a DLL and install it on the server. Then, your ASP page can call the DLL as needed, so the full functionality is available without exposing your source code to others.

In VB, you create a DLL by selecting ActiveX DLL from the New Project dialog box. The new project will contain a single class module, which illustrates how VB DLLs work. A DLL that you create will contain one or more classes, which are essentially identical to the classes you can create in a standard VB EXE project. When a program calls a DLL, it will actually be making use of the DLL's classes and their methods and properties. An ActiveX DLL can contain forms and other components as well, but classes are at the heart of their functionality.

Self Learning Exercise-2

i. Fill in the Blanks

b) The three types of ActiveX components are ____________,____________ and ___________.

c) To build a component that can run in the same process with your application, build an ActiveX ________.

d) ________.dll is Multiple Provider Router library.

ii. State True/False

b) Visual Basic ActiveX controls are based on the Visual Basic UserControl object.

c) You cannot create animation by displaying successive frames of the animation sequence at intervals.

d) When you call any DLLs directly from Visual Basic, you lose the built-in safety features of the Visual Basic environment.

7.8 Multimedia and VB6.0

Multimedia refers to devices other than the screen or the printer to play sounds, watch videos or record music. This is done through the use of the Multimedia control. Don't look for it in the toolbox, it's not there. It is an additional control that you must load.

First, create a new form in Project Lesson7 and call it "multimed.frm". Then, in the menu, at Project —> Components, find the item "Microsoft Multimedia Control 6.0" and check the box next to it. Hit OK and that's it. The Multimedia control should now appear in your toolbox.

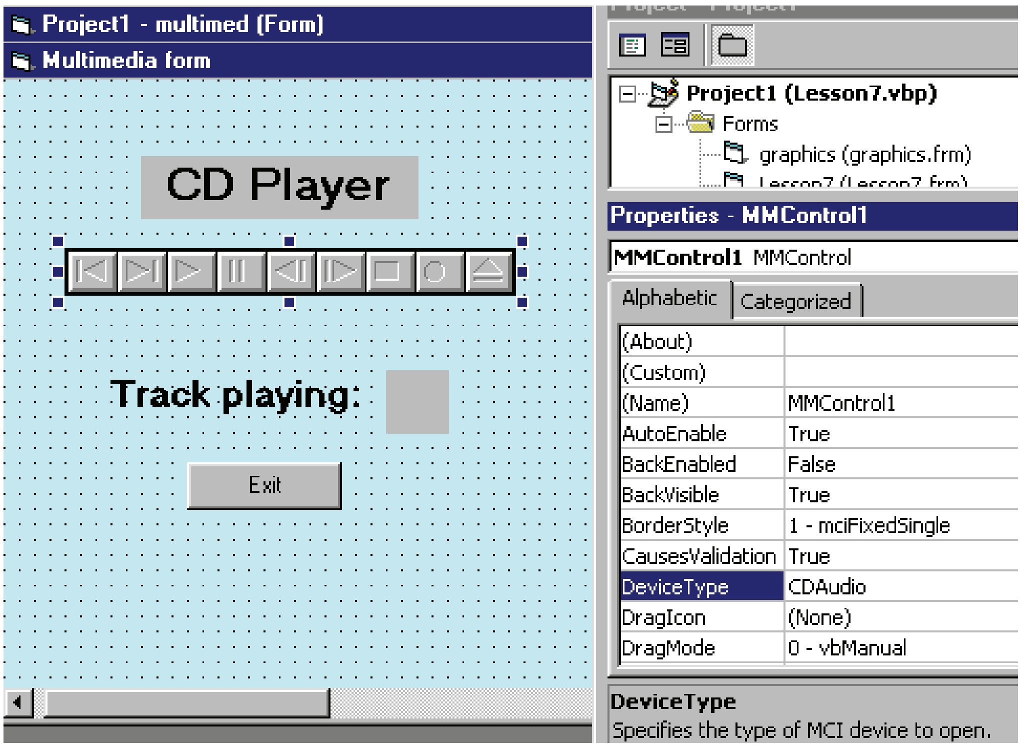
If you select the multimedia control and put it down on the form, you will have a button bar like all the ones you've seen on CD players, recorders, etc. In the DeviceType property you specify what type of device this control controls:

Table 7.2: Device Types

Device Type	Device
CDAudio	CD Audio Player
DAT	Digital Audio Tape Player
Overlay	Overlay
Scanner	Scanner
VCR	Video Tape Player and Recorder
Video Disc	Video Disk Player
Other	Other device not specified

7.8.1 Example: A Simple CD Player

We create a new form and call it multimed.frm. After adding the Multimedia control to the toolbox, we put a MM control on the form. Since we will only be using one MM control, we'll leave its name as MMControl1. The only property we have to change at this time is the DeviceType, to tell it that we are using the CD player, so we write CDAudio in DeviceType. We add a few labels to complete the form and we get:



Project1 (Lesson7.vbp)
Forms
graphics (graphics.frm)
Lesson7 (Lesson7.frm)

Properties - MMControl1

MMControl1 MMControl

Alphabetic | Categorized

(About)	
(Custom)	
(Name)	MMControl1
AutoEnable	True
BackEnabled	False
BackVisible	True
BorderStyle	1 - mciFixedSingle
CausesValidation	True
DeviceType	CDAudio
DragIcon	(None)
DragMode	0 - vbManual

DeviceType
Specifies the type of MCI device to open.

Now we have to write the code to operate the CD player.

Before we start to write the code there are a few things to know about the MM control. There is a Track property which contains the number of the current track. But its most important property is called the Command property and it can take on several values that, in fact, operate the device.

Table 7.3: Values of Command Property

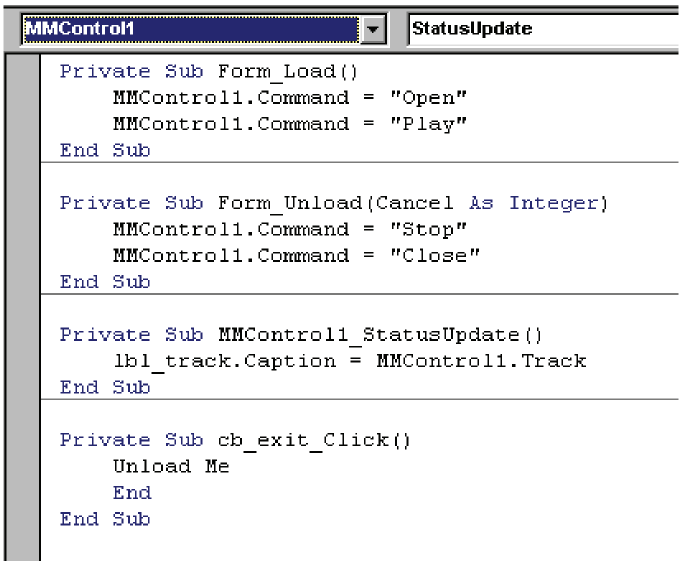
Command Value	Meaning
Open	Open the device
Close	Close the device
Eject	Eject the CD
Play	Play the device
Pause	Pause the device
Next	Go to next track
Prev	Go to beginning of the current track. If used within 3 second of most recent Prev, Goes to beginning of the previous track.
Record	Initialize recording
Save	Save open device files
Seek	Step backward and forward a track
Stop	Stop the device
Step	Step forward through tracks

Understand that both Track and Command are run-time properties because they are meaningless at design time. For example, to open the CD player:

MMControl1.Command = "Open" 'we assign the value "Open" to the Command property.
To pause: MMControl1.Command = "Pause" 'we assign the value "Pause" to the Command property.

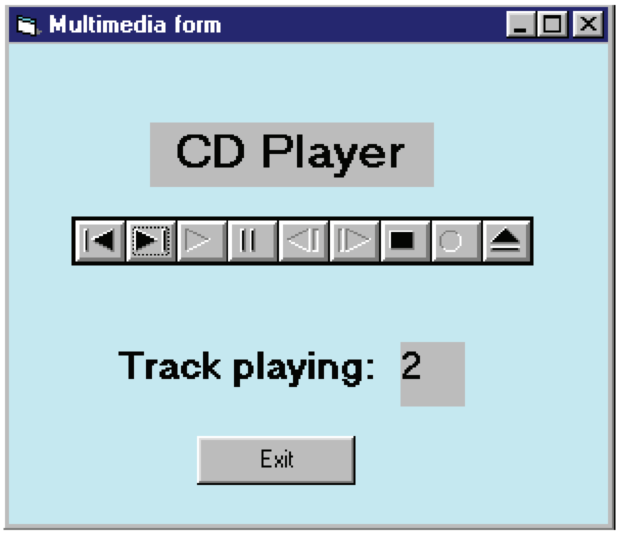
Now, as you have seen, the trick is to know with which events to associate the code that has to be written. The first one is fairly obvious: when we load the form, the Form_Load event, we will open the device. Now, one we haven't used before. When we unload the form, we will close the device. The reason is that, once launched, the device will keep on playing, even if the form is closed. So, just click on the Form_Unload event and write the code there. Finally, just to see that things are running smoothly, we will use the StatusUpdate event for the MM control to display the track being played every time the status of MMControl1 changes (change track, pause, play are all status changes).

As you will see, once the CD starts playing, you can use the buttons in the MM toolbar to control what it does.



```
Private Sub Form_Load()
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub

Private Sub MMControl1_StatusUpdate()
    lbl_track.Caption = MMControl1.Track
End Sub

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

The final application will look like this.



7.9 Concept of MFC Library

The Microsoft Foundation Class (MFC) Library is a collection of classes (generalized definitions used in object-oriented programming) that can be used in building application programs. The classes in the MFC Library are written in the C++ programming language. The MFC Library saves a programmer time by providing code that has already been written. It also provides an overall framework for developing the application program.

There are MFC Library classes for all graphical user interface elements (windows, frames, menus, tool bars, status bars, and so forth), for building interfaces to databases, for handling events such as messages from other applications, for handling keyboard and mouse input, and for creating ActiveX controls.

7.9.1 History

MFC was introduced in 1992 with Microsoft's C/C++ 7.0 compiler for use with 16-bit versions of Windows. It was part of an overall Microsoft effort to gain market share for development tools, and it was designed to showcase the capabilities of the C++ programming language. C++ was just beginning to replace C for development of commercial application software and C/C++ 7.0 was the first of Microsoft's compilers to add C++ support. MFC was inspired by, and owes much of its structure to, the Think Class Library (TCL) on Macintosh, later bought by Symantec.

MFC v8 was released with Visual Studio 2005. MFC is not included with the free Express edition of VS-2005. Microsoft recommends use of .NET for new development. The Object Windows Library (OWL), designed for use with Borland's Turbo C compiler, was a competing product introduced by Borland around the same time. Since it more strictly followed some OO design guidelines, OWL was more popular than MFC for a time. However, it lost market share when OWL updates lagged the addition of new features to Windows. Borland then released VCL (Visual Component Library) to replace the OWL framework. VCL is incompatible with OWL. A decision by Borland was made to discontinue OWL development and Borland began licensing MFC from Microsoft.

7.9.2 Advantages

Advantages of using MFC Library are

- It Provides an object-oriented programming model to the Windows APIs.

- C++ wrapper types for many common Windows resource-related data types that provide automatic closure of handles when the objects creating them go out of scope.
- Provides a Document/View framework for creating Model-View-Controller-based architectures.
- Provides utility classes such as CString and collection classes, which are usable even by console applications.
- Faster executables than produced by interpreters like Visual Basic.
- Generally faster executables than those produced by the .NET Framework.
- Capable IDE (integrated development environment) for debugging and code development.
- Relatively small GUI executables.
  - mfc42.dll has been included with all distributions of the Windows operating system since Win98-First Edition.
  - Windows 95 included mfc40.dll.
  - executables do not require the end-user to have the .NET Framework, which is over 15 mb.
  - alternatives to MFC either must link with that vendor's library, or supply a .dll.

Numerous wizards are available for a variety of tasks, especially generating minimal working applications from which to build.

- Integration with form designer to position controls.
- Large amount of resources available, from books to websites to sample code
- Fast compilation speed.
- Similar code base (but typically not completely the same) can be used for desktop and Pocket PC applications.

7.9.3 Disadvantages

Disadvantages of MFC Library are

- Minimally portable to other operating systems.
  - Mainsoft has made MFC tools available for Unix,
  - Microsoft shipped MFC versions for the Apple Macintosh operating systems in the 1990s. Visual Studio support for the Macintosh has since been discontinued.
  - Desktop and Pocket PC versions are similar, but not completely compatible.
- Depending on the specific situation, code size will generally be larger than using native Win32.
- Depending on the specific situation, performance can potentially be somewhat slower than using Win32.
- Large library with a significant learning curve.
- Alternatives such as VCL and Visual Basic provide RAD (rapid application development).
- Compatibility with past versions results in non-conformance with modern C++ standards.

- Microsoft recommends that new development use .NET, so the future of MFC is cloudy.
- It is not clear to what extent Microsoft actually used MFC for internal development of the software they market. Microsoft reportedly used vanilla C (along with a specialized framework) for most or all of the development of their Office suite.
- Still requires varying degrees of knowledge of the underlying Windows API depending on how much non-MFC wrapped and custom Windows API code is needed.
- Does not provide a complete abstraction layer interface to the underlying Windows API, requiring custom Windows API code for those portions not covered by MFC.
- .NET does a much better job at providing an object oriented, complete, natural, and well organized API.

7.9.4 Future of MFC Library

Once promoted by Microsoft, emphasis on MFC has been eclipsed by a number of other technologies, especially the family of computer languages associated with the .NET Framework. C# and Visual Basic are often preferred for commercial software development because the complexity of C++ and MFC is often a concern.

Microsoft still officially supports MFC by developing new versions with each new version of Microsoft Visual Studio (except the free Express version does not support MFC). Vendors and computer programmers that have made a strategic commitment to C++ and the Windows platform continue to use MFC for new development.

7.9.5 Versions of MFC Library

Table 7.4: MFC Library Versions

Product Version	MFC Version
Microsoft C/C++	MFC 1.0
Visual C++ 1.0	MFC 2.0
Visual C++ 1.5	MFC 2.5
Visual C++ 2.0	MFC 3.0
Visual C++ 2.1	MFC 3.1
Visual C++ 2.2	MFC 3.2
Visual C++ 4.0	MFC 4.0
Visual C++ 4.1	MFC 4.1
Visual C++ 4.2	MFC 4.2
Pocket PC Enbedded 3.0	MFC 4.2
Visual C++ 5.0	MFC 4.21
Visual C++ 6.0	MFC 6.0
Visual C++ .NET 2002	MFC 7.0
Visual C++ .NET 2003	MFC 7.1
Visual C++ 2005	MFC 8.0

7.10 Object Oriented Features of Visual Basic

Visual Basic 6.0 supports object-oriented language elements and has support for objects distributed in libraries. Visual Basic 2005 extends the support for object-oriented programming by supporting all of its language properties.

Access Levels

In Visual Basic 6.0, the keywords Private, Friend, Public, and Static are used to set access levels for declared elements.

In Visual Basic 2005, the keywords Private, Friend, Public, and Static, plus the new keywords Protected and Protected Friend are used to set access levels for declared elements. The access level of a declared element is the extent of the ability to access it— that is, what code has permission to read it or write to it.

Attributes

In Visual Basic 6.0, limited support for embedded attributes is provided through tools such as the Procedure Attributes in the Visual Basic IDE.

In Visual Basic 2005, an Attribute is a descriptive tag that can be used to annotate type and type member statements, thereby modifying their meaning or customizing their behavior. For example, class statements and class method statements can be annotated with attributes. Your application and other applications, such as the Visual Basic compiler, can use reflection to access attributes to determine how types and type members can be used.

Attributes can be used to perform Aspect Oriented Programming (AOP) with Visual Basic. An aspect is a part of a program that cross-cuts its business logic. In other words, it is needed to complete the program, but is not necessarily specific to the domain that the program is written for. Isolating such aspects as logging and persistence from business logic is the aim of the aspect-oriented programming paradigm.

Binary Compatibility

In Visual Basic 6.0, the Binary Compatibility option allows you to automatically retain class and interface identifiers from a previous version of a component when you compile a new version.

In Visual Basic 2005, the Binary Compatibility option is not supported; instead binary compatibility is accomplished with attributes. This gives you direct control over the information placed in your compiled component, such as class and interface identifiers, virtual table offsets, and any appropriate COM attributes.

Class Modules

In Visual Basic 6.0, a class is defined in a class module. A single class module is stored in a special type of file that has a .cls file extension.

In Visual Basic 2005, a class is defined in a Class statement that specifies the name and the members of a class. Class statements are stored in source files. The entire source file can be viewed as plain text. Multiple Class statements as well as other type statements can be stored in single source file. Visual Basic does not require the name of the source file to match a Class or type defined in the source file.

Class Constructor Methods

In Visual Basic 6.0, the class Initialize event handler named Class_Initialize is used to execute code that needs to be executed at the moment that an object is created.

In Visual Basic 2005, one or more constructors are added to a class to execute code and initialize variables. Constructors are the methods in a class that are named New. The New method can be overloaded to provide multiple constructors with the name New within the same class statement.

Class Destructor Methods

In Visual Basic 6.0, the Nothing keyword is used to disassociate an object variable from an actual object. The Set statement is used to assign Nothing to an object variable.

In Visual Basic 2005, for the majority of the objects that your application creates, you can rely on the garbage collector to automatically perform the necessary memory management tasks. However, unmanaged resources require explicit cleanup. The most common type of unmanaged resource is an object that wraps an operating system resource, such as a file handle, window handle, or network connection.

Class Initialization

In Visual Basic 6.0, a class Initialize event can be handled by a Class_Initialize method to execute code that needs to be executed at the moment an object is created. For example, the values of class data variables can be initialized.

In Visual Basic 2005, The Initialize event and the Class_Initialize handler are not supported. To provide class initialization, add one or more constructor methods to the classes and structures you define.

Data Classes

In Visual Basic 6.0, data source and complex data consumer classes are used to work with external data such as Microsoft SQL Server databases. A data source class provides data from an external source. A data consumer class can be bound to an external source of data such as a Data Source class.

In Visual Basic 2005, data source, simple data consumer, complex data consumer, and binding classes are used to work with external and internal data.

Default Property

In Visual Basic 6.0, any class property can be defined as the default property of the class.

In Visual Basic 2005, the default member of a class or structure can only be a property that takes one or more arguments. Default property members are defined by including the Default keyword in one property declaration statement in a class or structure.

Error Handling

In Visual Basic 6.0, the On Error statement is used for unstructured exception handling.

In Visual Basic 2005, both unstructured and structured exception handling are supported. Structured exception handling is a control structure containing exceptions, isolated blocks of code, and filters to create an exception-handling mechanism.

Events

In Visual Basic 6.0, the Event, RaiseEvent, and WithEvents keywords are used to declare, raise, and handle events.

In Visual Basic 2005, the Event, RaiseEvent, and WithEvents keywords plus the new AddHandler, RemoveHandler, and Handles keywords are used to declare, raise, and handle events. The AddHandler and RemoveHandler keywords allow you to dynamically add, remove, and change the event handler associated with an event. The Handles keyword allows you to define a Handles clause on a method. Handles declares that a procedure handles a specified event.

Generics

In Visual Basic 6.0, generic types are not supported.

In Visual Basic 2005, generics are used to implement parametric polymorphism in a Visual Basic program. Generic code is written without mention of any specific type and thus can be used transparently with any number of new types.

Global Classes

In Visual Basic 6.0, the value of the Instancing property of a class determines whether a class is private — that is, for use only within one component, or available for other applications to use. It also determines how other applications create instances of the class and how other applications call the class members.

In Visual Basic 2005, the Instancing property is no longer supported. You can apply the Public keyword to class statements in an assembly to expose classes in that assembly to other assemblies. You can reference an external assembly such as a class library to enable code in your application to use the Public classes from that class library.

You can use the Imports keyword to import namespace names from referenced projects and assemblies. The Imports keyword can also import namespace names defined within the same project as the file in which the statement appears.

You can apply the Shared keyword to field, property, and method members of classes and structures to implement shared members. Shared members are properties, procedures, and fields that are shared by all instances of a class or structure. The shared members of a class can be accessed without instantiating the class.

Implementation Inheritance

In Visual Basic 6.0, implementation inheritance is not supported.

In Visual Basic 2005, you can implement ad-hoc polymorphism through implementation inheritance. This allows you to define classes that can be used interchangeably by client code at run time, but with functionally different (yet identically named) methods or properties. You can define base classes that serve as the basis for derived classes. Derived classes inherit, and can extend, the properties, methods, and events of the base class. Derived classes can also override inherited methods with new implementations.

Interface Inheritance

Visual Basic 6.0 provides polymorphism through multiple ActiveX interfaces. In the Component Object Model (COM) that forms the infrastructure of the ActiveX specification, multiple interfaces allow systems of software components to evolve without breaking existing code. In Visual Basic 2005, you can implement ad-hoc polymorphism with the .NET Framework's Interface keyword.

Multiple classes may implement the same Interface, and a single class may implement one or more interfaces. Interfaces are essentially definitions of how a class needs to respond. An interface defines the methods, properties, and events that a class needs to implement, and the type of parameters each member needs to receive and return, but leaves the specific implementation of these members up to the implementing class. Multiple interfaces have the advantage of allowing systems of software components to evolve without breaking existing code.

Methods: Overloading

In Visual Basic 6.0, method overloading is not supported.

In Visual Basic 2005, Method overloading is used to implement ad-hoc polymorphism in a Visual Basic program. A method is overloaded when more than one version of the method is defined in a class. The overloaded versions differ in their parameters and returns types.

Methods: Overriding

In Visual Basic 6.0, methods can not be overridden.

In Visual Basic 2005, you can use the Overrides keyword to override a method in a derived class to provide a different implementation of a method from the derived class's base class.

Operator Overloading

In Visual Basic 6.0, operator overloading is not supported.

In Visual Basic 2005, operator overloading allows the behavior of a standard operator (such as *****, **<>**, or **And**) to be defined for a user-defined class or structure.

Variable Declaration

In Visual Basic 6.0, you can declare variables of different types in the same statement. If you do not specify the data type of a variable in the statement, it defaults to Variant.

In Visual Basic 2005, you can declare multiple variables of the same data type without having to specify the data type of each variable in the statement.

Object Lifetime

In Visual Basic 6.0, object lifetime is deterministic; every object instance maintains a reference count. When the last reference to an instance is released and the count equals zero, the object is terminated immediately.

In Visual Basic 2005, object lifetime is non-deterministic; a destructor is not necessarily called as soon as the last reference is released. This is because the common language runtime maintains a reference tree instead of individual reference counts. The garbage collector traces the reference tree in the background. If it finds an object or group of objects that have no references from any currently executing code, it calls the destructors of all such objects.

Reflection

In Visual Basic 6.0, reflection is not supported.

In Visual Basic 2005, the classes in the .NET Framework class library System. Reflection namespace can be used to obtain information about types such as classes, interfaces, and value types at run time and to create type instances to invoke and access them.

Self Learning Exercise-3

i. Fill in the Blanks

e) In Visual Basic 2005, the keywords Private, Friend, Public, and Static, plus the new keywords ________ and ______________ are used to set access levels for declared elements.

f) In Visual Basic 6.0, method overloading is _____ supported.

ii. State True/False

(e) In Visual Basic 6.0, methods can be overridden.

(f) DLL stands for Digital Link Library.

7.11 Summary

This unit gives an idea about dragging and dropping multiple objects and discusses graphics in Visual Basic in detail.

Simple animation can be created by displaying successive frames of the animation sequence at intervals in Visual Basic.

ActiveX controls and ActiveX documents are two of the ActiveX components, which can be built with Visual Basic.

Dynamic Link Library is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer.

The Microsoft Foundation Class (MFC) Library is a collection of classes (generalized definitions used in object-oriented programming) that can be used in building application programs.

Visual Basic also supports many features of object oriented programming like overloading, overriding, interface inheritance etc.

7.12 Glossary

ActiveX	ActiveX components are really OLE servers**.** There are three types of ActiveX Components : ActiveX Controls, ActiveX Documents and Code Components (OLE automation servers).
DLL	A dynamic link library (DLL) is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer.
MFC Library	The Microsoft Foundation Class (MFC) Library is a collection of classes (generalized definitions used in object-oriented programming) that can be used in building application programs.

7.13 Further Readings

1. Alan Eliason, Ray Malarkey, "Visual Basic 6, Environment, Programming and Application".
2. Brian Siller, Jeff Spotts, "Visual Basic 6".

7.14 Answers to Self Learning Exercises

i. Fill in the Blanks	ii. True/False
a) DoDragDrop	a) True
b) ActiveX Controls, ActiveX Documents, Code Components	b) True
c) Class Libraries	c) False
d) Mpr.dll	d) True
e) Protected and Protected friend	e) False
f) Not	f) False

7.15 Unit End Questions

1. Which are the different types of Windows DLL?
2. What is method overloading?
3. Explain the Visual Basic coordinate system.
4. Which method handles the class initialize event in VB6.0?
5. What is method overriding?
6. Describe the Layering.
7. Describe the MFC and DLL's.
8. Is VB object oriented? If yes, explain its features.
9. Explain the disadvantages of using MFC.
10. Explain ActiveX.

Unit - 8: VB.NET

Structure of the Unit

8.0 Objectives

After reading this chapter you will be able to understand,

- The basic concepts and features of VB .NET.
- Visual studio .NET IDE and Framework of .NET
- Concepts of Namespaces, Versioning and Attributes
- Migration from Visual Basic to Visual Basic .NET

8.1 Introduction

Visual Basic .NET provides the easiest, most productive language and tool for rapidly building Windows and Web applications. Visual Basic .NET comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices. We discuss the features of VB.Net in detail in the following sections.

Windows-based Applications

Visual Basic .NET comes with features such as a powerful new forms designer, an in-place menu editor, and automatic control anchoring and docking. Visual Basic .NET delivers new productivity features for building more robust applications easily and quickly.

Building Web-based Applications

With Visual Basic .NET we can create Web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. We can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages.

Easy Deployment

With Visual Basic .NET we can build applications more rapidly and deploy and maintain them with efficiency. Side-by-side versioning enables multiple versions of the same component to live safely on the same machine so that applications can use a specific version of a component. XCOPY-deployment and Web auto-download of Windows-based applications combine the simplicity of Web page deployment and maintenance with the power of rich, responsive Windows-based applications.

Power, Flexible, Simplified Data Access

We can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Correct access to ADO enables simple data access for connected data binding scenarios.

Full Object Oriented Construct

We can create reusable, enterprise-class code using full object-oriented constructs of VB.NET. Language features include full implementation inheritance, encapsulation, and polymorphism. Structured exception handling provides a global error handler.

XML Web Services

XML Web services enable us to call components running on any platform using open Internet protocols. Working with XML Web services is easier where enhancements simplify the discovery and consumption of XML Web services that are located within any firewall.

Mobile Application

Visual Basic .NET and the .NET Framework offer integrated support for developing mobile web applications for more than 200 Internet-enabled mobile devices. These new features give developers a single, mobile Web interface and programming model to support a broad range of Web devices, including WML 1.1 for WAP—enabled cellular phones, compact HTML (cHTML) for i-Mode phones, and HTML for Pocket PC, handheld devices, and pagers.

COM Interoperability

We can maintain our existing code without the need to reprogram. COM interoperability enables us to leverage our existing code assets and offers flawless bi-directional communication between Visual Basic 6.0 and Visual Basic .NET applications.

Reusability of Existing Investment

We can reuse all our existing ActiveX Controls. Windows Forms in Visual Basic .NET provide a robust container for existing ActiveX controls. In addition, full support for existing ADO code and data binding enable a smooth transition to Visual Basic .NET.

8.2 The Visual Studio.Net IDE

Visual Studio .NET IDE (Integrated Development Environment) is the Development Environment for all .NET based applications which comes with rich features. Visual Studio .NET IDE provides many options and is packed with many features that simplify application development by handling the complexities. Visual Studio .NET IDE is an enhancement to all previous IDE's by Microsoft.

The key features of Visual Studio .NET IDE are,

Common IDE for all .NET Projects

Visual Studio .NET IDE provides a single environment for developing all types of .NET applications. Applications range from single window applications to complex n-tier applications and rich web applications.

Option to choose from Multiple Programming Languages

We can choose the programming language of our choice to develop applications based on our expertise in that language. We can also incorporate multiple programming languages in one .NET solution and edit that with the IDE.

Customizable

We can customize the IDE based on our preferences. The 'My Profile' settings allow us to do this. With these settings we can set the IDE screen the way we want, the way the keyboard behaves and we can also filter the help files based on the language of our choice.

Built-in Browser

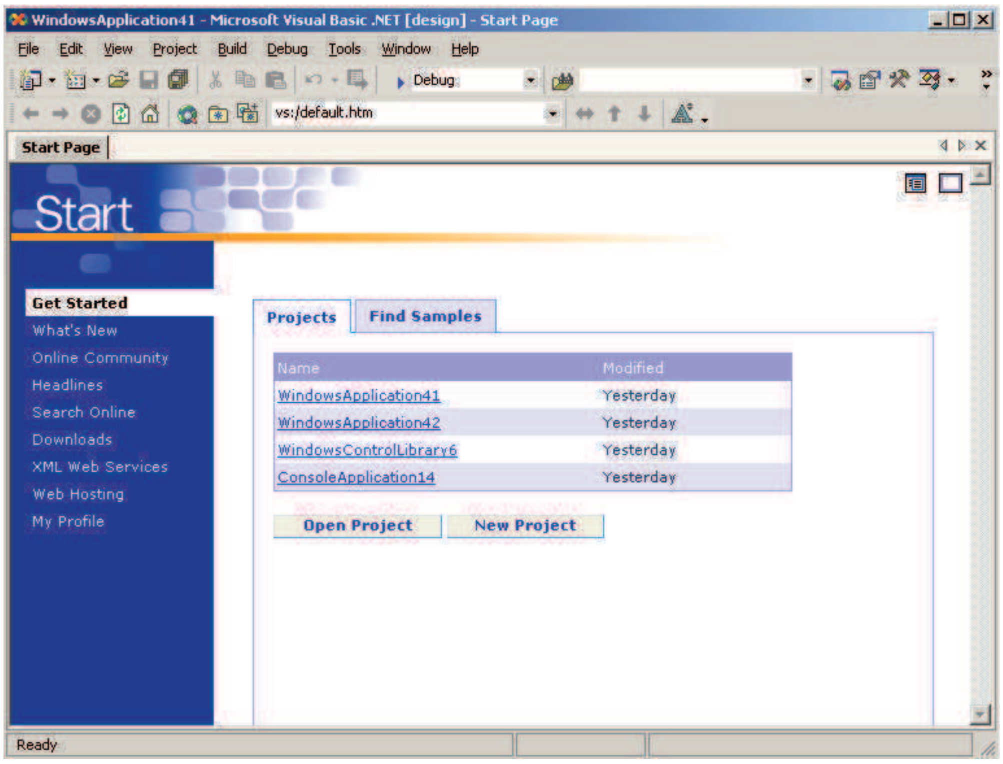
The IDE comes with a built-in browser that helps us to browse the Internet without launching another application. We can look for additional resources, online help files, source codes and much more with this built-in browser feature.

Working with Visual Studio .NET IDE

Start Page

When we open VS .NET from

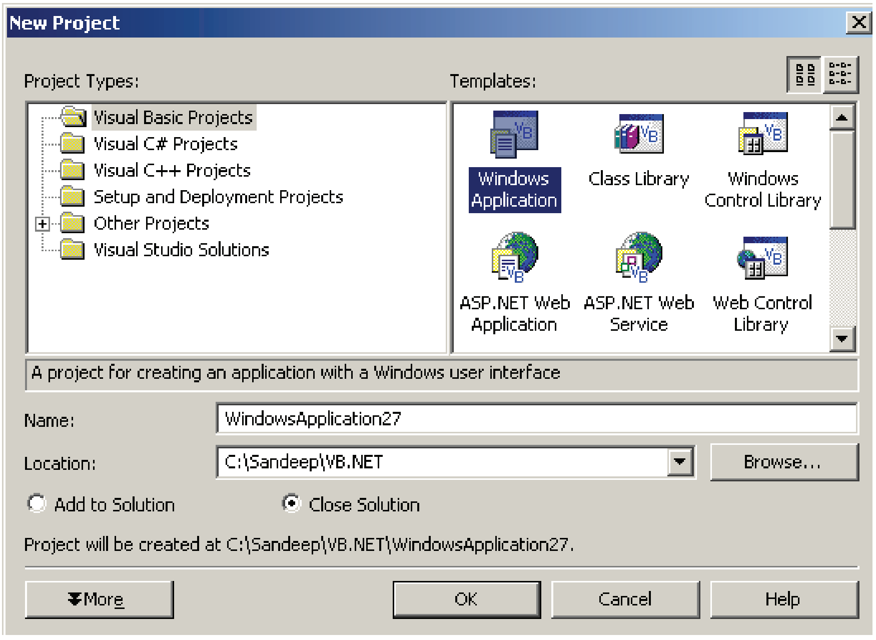
Start->Programs->Microsoft Visual Studio .NET->Microsoft Visual Studio .NET, the Start Page appears in the main area of the screen. The start Page allows us to select from the most recent projects (last four projects) with which we worked or it can be customized based on our preferences.



Name	Modified
WindowsApplication41	Yesterday
WindowsApplication42	Yesterday
WindowsControlLibrary6	Yesterday
ConsoleApplication14	Yesterday

New Project Dialogue Box

The New Project dialogue box like the one in the image below is used to create a new project specifying it's type allowing us to name the project and also specify it's location where it is saved. The default location on the hard disk where all the projects are saved is C:\DocumentsandSettings\Administrator\MyDocuments\VisualStudioProjects.



A project for creating an application with a Windows user interface

Name: WindowsApplication27

Location: C:\Sandeep\VB.NET

Add to Solution [✓ Close Solution]

Project will be created at C:\Sandeep\VB.NET\WindowsApplication27.

Following are different templates under Project Types and their use.

Windows Application

This template allows creating standard windows based applications.

Class Library

Class libraries are those that provide functionality similar to ActiveX and DLL by creating classes that access other applications.

Windows Control Library

This allows to create our own windows controls. Also called as User Controls, where you group some controls, add it to the toolbox and make it available to other projects.

ASP .NET Web Application

This allows to create web-based applications using IIS. We can create web pages, rich web applications and web services.

ASP .NET Web Service

Allows to create XML Web Services.

Web Control Library

Allows to create User-defined controls for the Web. Similar to user defined windows controls but these are used for Web.

Console Application

A new kind of application in Visual Studio .NET. They are command line based applications.

Windows Service

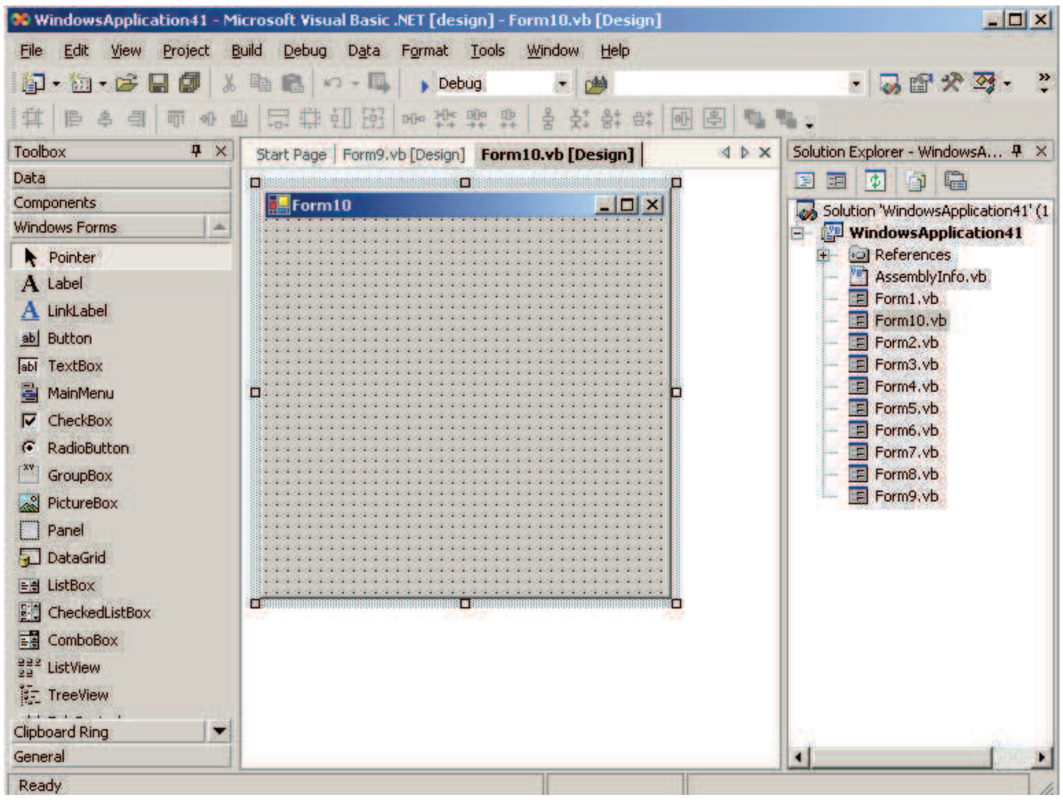
These run continuously regardless of the user interaction. They are designed for special purpose and once written, will keep running and come to an end only when the system is shut down.

Other

This template is to develop other kinds of applications like enterprise applications, database applications etc

Working Window

The Integrated Development Environment (IDE) shown in the image below is what we actually work with. This IDE is shared by all programming languages in Visual Studio. You can view the toolbars towards the left side of the image along with the Solution Explorer window towards the right.



Self Learning Exercise-1

i. Fill in the Blanks

a) ASP .NET Web Application allows to create ________ applications using IIS.

b) _____________are command line based applications.

ii. State True/False

a) IDE stands for Interface Development Environment

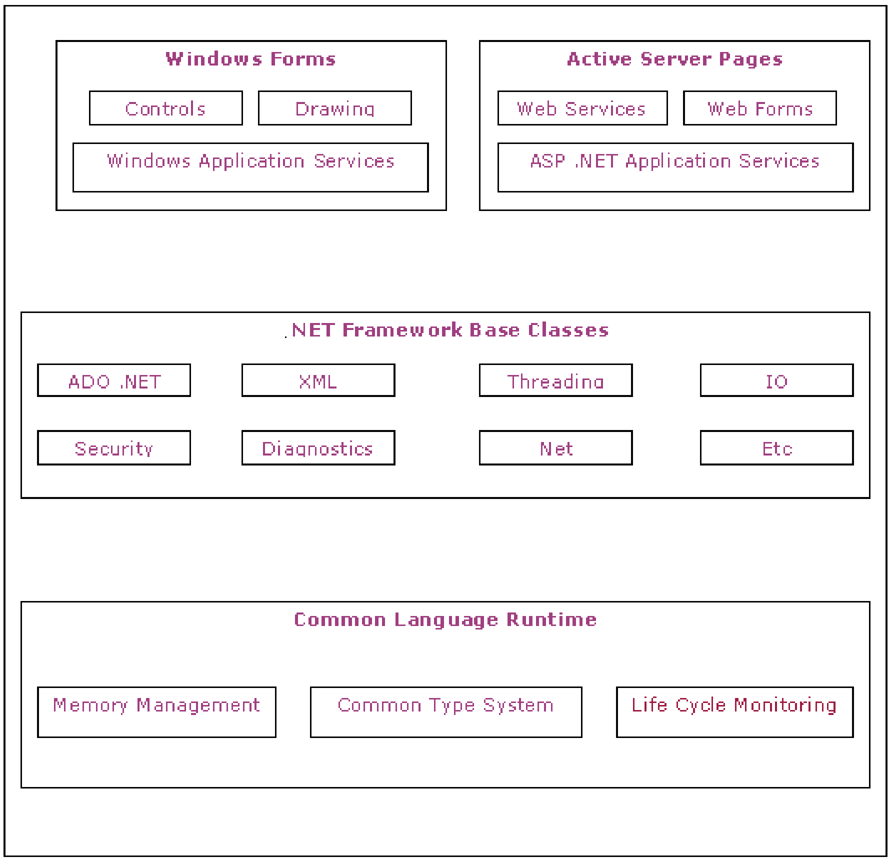
b) ASP .NET Web Service allows to create windows Services

8.3 The .NET Framework

.NET is a "Software Platform". It is a language-neutral environment for developing rich .NET experiences and building applications that can easily and securely operate within it. When developed applications are deployed, those applications will target .NET and will execute wherever .NET is implemented instead of targeting a particular Hardware/OS combination. The components that make up the .NET platform are collectively called the .NET Framework.

The .NET Framework is a managed, type-safe environment for developing and executing applications. The .NET Framework manages all aspects of program execution, like, allocation of memory for the storage of data and instructions, granting and denying permissions to the application, managing execution of the application and reallocation of memory for resources that are not needed.

The .NET Framework is designed for cross-language compatibility. Cross-language compatibility means, an application written in Visual Basic .NET may reference a DLL file written in C# (C-Sharp). A Visual Basic .NET class might be derived from a C# class or vice versa.



8.3.1 Components of .NET Framework

The .NET Framework consists of two main components:

Common Language Runtime (CLR)

Class Libraries

Common Language Runtime(CLR)

The CLR is described as the "execution engine" of .NET. It provides the environment within which the programs run. This CLR manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the Common Type System (CTS), it enforces strict type safety, and it ensures that the code is executed in a safe environment by enforcing code access security. The software version of .NET is actually the CLR version.

Working with CLR

When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL), which is a low-level set of instructions understood by the common language run time. This MSIL defines a set of portable instructions that are independent of any specific CPU. It's the job of the CLR to translate this Intermediate code into a executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just In Time) complier. The process goes like this, when .NET programs are executed, the CLR activates the JIT complier. The JIT complier converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.

Class Libraries

Class library is the second major entity of the .NET Framework which is designed to integrate with the common language runtime. This library gives the program access to runtime environment. The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.

8.3.2 .NET Framework Advantage

The .NET Framework offers a number of advantages to developers. The following paragraphs describe them in detail.

Consistent Programming Model

Different programming languages have different approaches for doing a task. For example, accessing data with a VB 6.0 application and a VC++ application is totally different. When using different programming languages to do a task, a disparity exists among the approach developers use to perform the task. The difference in techniques comes from how different languages interact with the underlying system that applications rely on.

With .NET, for example, accessing data with a VB .NET and a C# .NET looks very similar apart from slight syntactical differences. Both the programs need to import the System. Data namespace; both the programs establish a connection with the database and both the programs run a query and display the data on a data grid. .NET example explains that there is a unified means of accomplishing the same task by using the .NET Class Library, a key component of the .NET Framework.

The functionality that the .NET Class Library provides is available to all .NET languages resulting in a consistent object model regardless of the programming language the developer uses.

Direct Support for Security

Developing an application that resides on a local machine and uses local resources is easy. In this scenario, security is not an issue as all the resources are available and accessed locally. Consider an application that accesses data on a remote machine or has to perform a privileged task on behalf of a no privileged user. In this scenario security is much more important as the application is accessing data from a remote machine.

With .NET, the Framework enables the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. This makes distributed computing more secure because .NET developers cooperate with network security devices instead of working around their security limitations.

Simplified Development Efforts

Let’s take a look at this with Web applications. With classic ASP, when a developer needs to present data from a database in a Web page, he is required to write the application logic (code) and presentation logic (design) in the same file. He is required to mix the ASP code with the HTML code to get the desired result.

ASP.NET and the .NET Framework simplify development by separating the application logic and presentation logic making it easier to maintain the code. You write the design code (presentation logic) and the actual code (application logic) separately eliminating the need to mix HTML code with ASP code. ASP.NET can also handle the details of maintaining the state of the controls, such as contents in a textbox, between calls to the same ASP.NET page.

Another advantage of creating applications is debugging. Visual Studio .NET and other third party providers provide several debugging tools that simplify application development. The .NET Framework simplifies debugging with support for Runtime diagnostics. Runtime diagnostics helps you to track down bugs and also helps you to determine how well an application performs. The .NET Framework provides three types of Runtime diagnostics: Event Logging, Performance Counters and Tracing.

Easy Application Deployment and Maintenance

The .NET Framework makes it easy to deploy applications. In the most common form, to install an application, all you need to do is copy the application along with the components it requires into a directory on the target computer. The .NET Framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET Framework ensures that all the components the application depends on are available on the computer before the application begins to execute.

8.3.3 .NET Framework Supported Languages

As mentioned on the .NET Framework introduction, .NET Framework is designed for cross-language compatibility. Cross-language compatibility means .NET components can interact with each other irrespective of the languages they are written in. An application written in VB .NET can reference a DLL file written in C# or a C# application can refer to a resource written in VC++, etc. This language interoperability extends to Object-Oriented inheritance.

The table below lists all the languages supported by the .NET Framework and describes those languages. The languages listed below are supported by the .NET Framework up to the year 2003. In future there may be other languages that the .NET Framework might support.

Table 8.1 Languages Supported by .NET Framework

Languages	Description/Usage
APL	APL is one of the most powerful, consistent and concise computer programming languages ever devised. It is a language for describing procedures in the processing of information. It can be used to describe mathematical procedures having nothing to do with computers or to describe the way a computer works
C++	C++ is a true OOP. It is one of the early Object-Oriented programming languages. C++ derives from the C language. Visual C++ is the name of a C++ compiler with an integrated environment from Microsoft. This includes special tools that simplify the development of great applications, as well as specific libraries. Its use is known as visual programming.
C#	C# called as C Sharp is a full-fledged Object-Oriented programming language from Microsoft built into the .NET Framework. First created in the late 1990's was part of Microsoft's whole .NET strategy.

COBOL	COBOL (Common Business Oriented Language) was the first widely used high-level programming language for business applications. It is considered as a programming language to have more lines of code than any other language.
Component Pascal	Component Pascal is a Pascal derived programming language that is specifically designed for programming software components.
Eiffel	Eiffel is an Object-Oriented (OO) programming language, which emphasizes the production of robust software. Eiffel is strongly statically typed mature Object-Oriented language with automatic memory management.
Forth	Forth is a programming language and programming environment. It features both interactive execution of commands (making it suitable as a shell for systems that lack a more formal operating system), as well as the ability to compile sequences of commands into threaded code for later execution.
ForTran	Acronym for Formula Translator, Fortran is one of the oldest high-level programming languages that is still widely used in scientific computing because of its compact notation for equations, ease in handling large arrays, and huge selection of library routines for solving mathematical problems efficiently.
Haskell	Haskell is a computer programming language that is a polymorphicly typed, lazy, purely functional language, quite different from most other programming languages. It is a wide-spectrum language, suitable for a variety of applications. It is particularly suitable for programs which need to be highly modifiable and maintainable.
Java Language	The Java language is one of the most powerful Object-Oriented programming languages developed till date. It's platform independence feature makes it a very popular programming language.
Microsoft JScript	Microsoft JScript is the Microsoft implementation of the ECMA 262 language specification. JScript is an interpreted, object-based scripting language. It has fewer capabilities than full-fledged Object-Oriented languages like C++ but is more than sufficiently powerful for its intended purposes.
Mercury	Mercury is a new logic/functional programming language, which combines the clarity and expressiveness of declarative programming with advanced static analysis and error detection features.
Oberon	Oberon is a programming language very much like Modula-2 in syntax but with several interesting features. It's based on OOP concepts and provides a Windows-based graphical user interface.

Oz	Oz is a high-level programming language that combines constraint inference with concurrency. Oz is dynamically typed and has first-class procedures, classes, objects, exceptions and sequential threads synchronizing over a constraint store
Pascal	Principle objectives for Pascal were for the language to be efficient to implement and run, allow for the development of well structured and well organized programs, and to serve as a vehicle for the teaching of the important concepts of computer programming. The Prime area of application that Pascal entails is the learning environment
Perl	Practical Extraction and Report Language, Perl, is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks.
Python	Python is an interpreted, interactive, Object-Oriented programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high-level dynamic data types, and dynamic typing.
RPG	Report Program Generator, RPG, is used for generation of reports from data files, including matching record and sub-total reports. RPG is one of the few languages created for punch card machines that is still in common use today
Scheme	Scheme is a statically scoped programming language. It was designed to have an exceptionally clear and simple semantics and few different ways to form expressions
Small Talk	Smalltalk is an expressive language that uses a simple sub set of human languages, nouns and verbs. Smalltalk was the first, and remains one of the few, pure object systems, which simply means that everything in a Smalltalk program is an object. Smalltalk is generally recognized as the second Object Programming Language (OPL).
Standard ML	Standard ML is a safe, modular, strict, functional, polymorphic programming language with compile-time type checking and type inference, garbage collection, exception handling, immutable data types and updateable references, abstract data types, and parametric modules
Microsoft Visual Basic	Visual Basic is a "visual programming" environment for developing Windows applications. Visual Basic makes it possible to develop complicated applications very quickly. This site is all about Visual Basic.

Self Learning Exercise-2

i. Fill in the Blanks

c) The .NET Framework consists of two main components: CLR and _________.

d) ____________ means .NET components can interact with each other irrespective of the languages they are written in.

ii. State True/False

c) CLR stands for Common Language Runtime.

d) Class library is the second major entity of the .NET Framework which is designed to integrate with the common language runtime.

8.4 VB.NET and Internet

The Internet is a network of networks, linking computers to computers sharing the TCP/IP protocols. Each runs software to provide or "serve" information and/or to access and view information. The Internet is the transport vehicle for the information stored in files or documents on another computer. It can be compared to an international communications utility servicing computers. It is sometimes compared to a giant international plumbing system. The Internet itself does not contain information. It is a slight misstatement to say a "document was found *on* the Internet." It would be more correct to say it was found through or using the Internet. What it was found in (or on) is one of the computers linked to the Internet.

Internet may use one or all of the following Internet services:

Electronic mail (e-mail):

Permits you to send and receive mail.

Telnet or remote login:

Permits your computer to log onto another computer and use it as if you were there.

FTP or File Transfer Protocol:

Allows your computer to rapidly retrieve complex files intact from a remote computer and view or save them on your computer.

The World Wide Web (WWW or "the Web"):

The largest, fastest growing activity on the Internet.

To make web application development easier and faster VB.NET architecture provides us many useful tools and facilities With VB.NET we can create web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. You can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages. . The main features of VB.NET, which make it a right choice for web application development, are already discussed in introduction.

8.4.1 XML Web Services & Mobile Application Development

VB.NET also provides XML web services. These services enable us to call components running on any platform using open Internet protocols. Working with XML Web services is easier where enhancements simplify the discovery and consumption of XML Web services that are located within any firewall. Mobile and PDA's are becoming the most popular devices to access Internet. That's why VB.NET and the .NET Framework offer integrated support for developing mobile web applications for more than 200 Internet-enabled mobile devices. These new features give developers a single, mobile Web interface and programming model to support a broad range of Web devices, including WML 1.1 for WAP—enabled cellular phones, compact HTML (cHTML) for i-Mode phones, and HTML for Pocket PC, handheld devices, and pagers.

8.4.2 Simplified Data Access

We can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Seamless access to ADO enables simple data access for connected data binding scenarios.

8.4.3 Security

Developing an application that resides on a local machine and uses local resources is easy. In this scenario, security isn't an issue as all the resources are available and accessed locally. Consider an application that accesses data on a remote machine or has to perform a privileged task on behalf of a no privileged user. In this scenario security is much more important as the application is accessing data from a remote machine.

VB.NET enables the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. This makes distributed computing more secure because .NET developers cooperate with network security devices instead of working around their security limitations.

8.4.4 Improved Coding Style

You can code faster and more effectively. A multitude of enhancements to the code editor, including enhanced IntelliSense, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a Rapid Application Development(RAD) coding machine.

Self Learning Exercise-3

i. Fill in the Blanks

e) RAD stands for ______________.

f) VB.NET provides ___________which enable us to call components running on any platform using open Internet protocols.

ii. State True/False

e) VB.NET enables the developer and the system administrator to specify method level security

8.5 Migration from VB to VB.NET

Microsoft Visual Basic has had many evolutions since its original release, Visual Basic 1.0. The release of Visual Basic .NET is the biggest evolution yet. The language has been redesigned to take advantage of the .NET Framework. By leveraging the features that the .NET Framework provides, Visual Basic supports language features such as code inheritance, visual forms inheritance, and multi-threading. The object model is more extensive than earlier versions, and Visual Basic .NET totally integrates with the .NET Framework. Therefore, interaction between components written in other .NET languages is very efficient.

Why migrate to VB.NET?

The first question that strikes ones mind is “why should I migrate?” Managed Code is a very important feature of the .NET architectural framework. This is what makes Java applications robust when compared to older generation Microsoft applications. VB.NET code targets the Common Language Runtime (CLR) by compiling into an intermediate language, which is then executed under strict control (managed). CLR manages the code in a very similar fashion to the Java model, thereby making applications much more Robust, Stable and secure. The application also becomes more maintainable because of the managed code.

Under the .NET framework, VB.NET applications are packaged as self-contained, self-describing, versioned assemblies (collection of classes and meta data). One can deploy, remove and manage N different versions of an application or DLL on the same machine without creating conflicts. This enables a product company to easily manage versions and upgrades without causing the headaches so common with usual upgrades or OEM-versioning. Desktop applications can be much more easily converted to Web Based applications using the Web Forms paradigm of .NET made available through VB.NET. Microsoft strategy is to eventually migrate all its products to the .NET framework This would mean that a couple of years down the line VB6, and other non-.NET products would be obsolete.

.NET applications when developed carefully can be platform independent. Managed code built using .NET will run on platforms that host the .NET framework irrespective of the Operating System or machine type. Such non-Microsoft platforms have started to appear already. If one is convinced of a necessity to migrate to VB.NET, the next question that arises is “How to migrate?”

8.5.1 Methods of Migrating

After getting a feel of VB.NET, let us now look at the methods to migrate the application from VB to VB.NET. The various methods that can be used to migrate the application are covered below. In many cases, these approaches will be combined. A redesign from scratch may be better for some code modules whereas many others might need only language/syntax level porting.

Migration Wizard

Visual Studio.NET provides a Visual Basic Upgrade Wizard (also called as Migration Wizard). This wizard does almost the entire migration barring some modifications that must be made to complete the upgrade process.

Fresh Design

Redesign of certain core modules or perhaps the whole application is a possible approach if new features of the .NET architectural framework are considered. Managed code (under CLR) can run in tandem with unmanaged code (e.g. COM components). Therefore it is possible to reuse many existing components as such. However, paradigm shifts towards XML web services, Web Forms, Windows Forms etc. call for some rework in design, coding and deployment.

Use of Customized tools to enhance migration

Porting driven by a wizard and re-design are easier when migrated using customized tools like the “Cal-Migrater” developed internally by CSWL for a health care product migration. This tool makes use of both the wizard and redesign approach to make faster pace through the .NET porting activity.

8.5.2 Migration Strategies

There are two strategies to migrate using the above methods viz. Vertical migration and Horizontal migration. Both these strategies help us find out key areas that need an immediate upgrade and the modules that can be upgraded later.

Vertical Migration

In vertical migration, a module of the application that is fairly standalone or isolated is taken, and all the tiers of this module are migrated to VB.NET. Vertical migration can be adopted in situations where the modules are mostly loosely coupled and/or we are not sure of the risk and effort factors of total migration. Thus migrating one module will give a good idea on the effort required to migrate other modules.

Horizontal Migration

In horizontal migration, one layer of all the modules is migrated to VB.NET followed by other layers. For example, first the UI would be migrated then the middle tier. Horizontal migration is ideal in situations where the modules are tightly coupled and the effort and risk involved in the migration is known to a large extent.

Issues While Migrating

However there are a few issues that are to be taken care of while migrating from VB to VB.NET. As mentioned earlier, porting the application using the wizard approach does automate the process to an extent but some modifications must be made to complete the upgrade process. The wizard produces a report in HTML format with the information of the places where it failed to migrate the application to .NET. There are number of changes that can be done to the VB6 application to make the upgrade process much more efficient. Some important aspects are listed below.

- Avoid using late binding. This is because properties and methods cannot be verified during the upgrade process.
- Specify default properties. VB6 specifies a default property for every component. For example, the default of Textbox is Text property and Caption is default for Label component. But in VB.NET there is nothing called as default property.
- Use Zero-Bound Array. In VB we can declare an array with any positive integer as its lower bound. But in VB.NET all the arrays are zero bound.
- Examine API calls with fixed length strings in VB application because VB.NET doesn’t support fixed length strings.
- Lines and Shapes are not supported in VB.NET and hence cannot be upgraded. Instead a graphic object is provided for shapes.
- Use constants instead of underlying values. VB6 constants will convert to the correct value when upgraded to VB.NET, but if actual values are used then it may end up with wrong hard-coded values. For example, for Boolean values using -1 and 0 instead of True and False and will have an adverse effect in VB.NET because in VB.NET True is1.
- Re-Design of the whole application can be considered if the application will Grow over time. VB.NET being object oriented, the code can be reused and the application can be extended with less effort and time. Doing the same in VB6 would need a lot more time & effort.

- Be taken online, i.e. web application. VB.NET provides web forms that can be used both for web application and VB application. This gives it dual manageability.

Self Learning Exercise-4

i. Fill in the Blanks

g) There are two strategies to migrate using the above methods: Vertical migration and _____________ .

h) In____________, one layer of all the modules is migrated to VB.NET followed by other layers.

ii. State True/False

f) VB.NET provides web forms that can be used both for web application and VB application.

8.6 Namespaces

A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace. The namespace with all the built-in VB functionality is the System namespace. All other namespaces are based on this System namespace.

A *namespace* can be seen as a container for some classes in much the same way that a folder on your file system contains files. Namespaces are needed because there are a lot of .NET classes. Microsoft has written many thousands of base classes, and any reasonably large application will define many more. By putting the classes into namespaces we can group related classes together, and also avoid the risk of name collisions: If your company happens to define a class that has the same name as the class written by another organization, and there were no namespaces, there would be no way for a compiler to figure out which class a program is actually referring to. With namespaces, there isn't a problem because the two classes will be placed in different namespaces, which compares with, say, the Windows files system where files with the same name can be contained in different folders.

It is also possible for namespaces to contain other namespaces, just as folders on your file system can contain other folders as well as files. **System** is the basic namespace used by every .NET code. If we can explore the System namespace little bit, we can see it has lot of subordinate namespaces. For example, System.IO, System.Net, System.Collections, System.Threading, etc.

8.6.1 Creating Namespaces

Every project in Visual Basic.Net has a root namespace, which is set in the Property page of the project. When we create a project, by default, the name of the root namespace for the project is set to the name of the new project. For example, the root namespace for a project named ProjectOne is ProjectOne. Usually, when we create a project, we would like to give it a name, which we like. So, we need to change the ProjectOne to the name which we wish. We can also organize classes using the Namespace keyword.

The Example

```
Namespace a.VB.Namespaces
Public Class Hello
Public Function GetMessage() As String
Return "Hello, world"
```

```
End Function ‘GetMessage
End Class ‘Hello
End Namespace ‘a.VB.Namespaces
```

This is simple namespace example. We can also build hierarchy of namespace. Here is an example for this.

```
Namespace Books
Namespace Inventory
Imports System
Class AddInventory
Public Function MethodOne()
Console.WriteLine(“Adding Inventory through MethodOne!”)
End Function
End Class
End Namespace
End Namespace
```

8.6.2 Using Namespaces in VB.NET

We have created a Namespace in the previous step. Now, we will use the Namespace explicitly through direct addressing or implicitly through the Imports statement.

Direct addressing involves directly accessing any class in the namespace by providing the fully qualified name.

Example of using fully qualified name is given below:

```
Microsoft.VisualBasic.MsgBox(“Using Fully Qualified Name”)
```

If we want to make all the classes in a given namespace available without the need to type the entire namespace each time, you can use the Imports statement. An example of using the Imports statement is given below:

```
Imports Microsoft.VisualBasic
…..
MsgBox(“Using Imports Statement”)
```

8.6.3 Types of Namespaces

Namespaces can be divided in to two categories:

Local Namespaces: These are accessible only to the applications to which they belong.

Global Namespaces: These are accessible from all applications.

Some Namespaces and Their Use

- System: Includes essential classes and base classes for commonly used data types, events, exceptions and so on.
- System.Collections: Includes classes and interfaces that define various collection of objects such as list, queues, hash tables, arrays, etc.
- System.Data: Includes classes which lets us handle data from data sources.

- System.Data.OleDb: Includes classes that support the OLEDB .NET provider.
- System.Data.SqlClient: Includes classes that support the SQL Server .NET provider.
- System.Diagnostics: Includes classes that allow to debug our application and to step through our code.
- System.Drawing: Provides access to drawing methods.
- System.Globalization: Includes classes that specify culture-related information.
- System.IO: Includes classes for data access with Files.
- System.Net: Provides interface to protocols used on the internet.
- System.Reflection: Includes classes and interfaces that return information about types, methods and fields.
- System.Security: Includes classes to support the structure of common language runtime security system.
- System.Threading: Includes classes and interfaces to support multithreaded applications.
- System.Web: Includes classes and interfaces that support browser-server communication.
- System.Web.Services: Includes classes that let us build and use Web Services.
- System.Windows.Forms: Includes classes for creating Windows based forms.
- System.XML: Includes classes for XML support..

Self Learning Exercise-5

i. Fill in the Blanks

i) ________is the basic namespace used by every .NET code.

j) Namespaces can be divided in to two categories: Local Namespaces and __________.

ii. State True/False

g) A namespace is a collection of different packages.

h) System.Data.OleDb includes classes that support the OLEDB .NET provider.

8.7 Versioning

The .NET Framework helps in building reliable applications by minimizing conflicts between components with a new versioning strategy involving public and private components. Private components reside in the same directory as the application using them. Applications don't consider component version when they're using private components; an application loads whatever physical file it finds in its directory.

But with public components, .NET applications always load the component version used when you created an application. Applications don't just use blindly whatever files the Registry points to. Using public components lets you install multiple versions of the same component on a machine. .NET also lets you modify how apps use public components.

To learn how to manage component versioning, begin by looking at .NET assemblies. Assemblies are files—usually DLLs—that contain one or more .NET classes and the metadata that describes them. Each assembly contains its own version number, along with the name, public key, and version of every assembly it references. This strategy forever links any assembly with all the assemblies it uses, including the version numbers.

When you create a new component, .NET stores the component's version number in the assembly's metadata. Every project includes the source file AssemblyInfo. The extension matches the project type. .NET automatically adds an Assembly Version attribute for your project in the source file. The version looks like this in VB projects:

<Assembly: AssemblyVersion("1.0.*")>

The C# version differs only superficially:

[assembly: AssemblyVersion("1.0.*")]

Version numbers have four components: major version, minor version, build number, and revision number. The default version 1.0.* sets the build number to the number of days since January 1, 2000, and the revision number to the number of seconds since midnight, local time. So the build and revision numbers increase over time. If you're creating commercial components, you'll want to change this default attribute to better manage the versions of your component in the field:

<Assembly: AssemblyVersion("1.0.0.0")>

Each build should increase the build number on the assembly version.

8.7.1 Assemblies

The 'Assembly' is a new concept that the .NET framework introduces to make your journey in programming more easier. The .NET framework introduces assemblies as the main building blocks of your application. An application can contains one or more assemblies. An assembly can be formed in one or more files. This all depends on your programming needs. An assembly can consist of the following four elements:

- Your code, compiled into MS Intermediate Language (MSIL). This code file can be either an EXE file or a DLL file.
- The assembly manifest, which is a collection of metadata that describes assembly name, culture settings, list of all files in the assembly, security identity, version requirements, and references to resources. The assembly manifest can be stored with the intermediate code, or in a standalone file that contains only assembly manifest information.
- Type metadata
- Resources

An assembly can have two types of versions. The first one which we call "Version Number" consists of a four-part string with the following format:

<major number>,<minor number>,<build>,<revision>

For example a version number of 3.5.20.1 indicates 3 as the major version, 5 as the minor version, 20 as the build number, and 1 as the revision number.

The second type of versions is called "Informational Version". The informational version consists of a string that contains the version number besides additional information like packaging, marketing literature, or product name. This type of version is used for informational purposes only, and is not used at runtime for calculating versioning related decisions.

8.7.2 .NET Version Number

The .NET version number consists of four parts:

<major number>,<minor number>,<build>,<revision>

Assume that you are working on a project and that you are building a new version on regular basis, Such as daily. Each day that you build the application, you will increment the build version.

If you have to make multiple builds in one day, you can increment the revision number. Regardless how you actually use the four parts of the version to track your project, They become very important when you begin changing them.

The first two numbers are often referred to as the *logical version number* of an assembly. For example, in the COM world, it is often common to ask, "What version of ADO you are using?" with common answer being "2.5" or "2.6."

Although the first two numbers might be the logical version, however, the client applications of strongly named assemblies care about all the numbers. The runtime sees assemblies with different version numbers as different assemblies, and this means your client, by default, will see these as different assemblies. It is actually possible for the administrator to override this default behavior if necessary.

8.7.3 Setting The Version Number

Setting the version number can be accomplished in a couple of ways. If you look in the AssemblyInfo.vb file that is created as a part of your project, you will see the following line:

```
<Assembly: AssemblyVersion("1.0.*")>
```

This will allow you to set the version for the application. Or, you can enter the following lines in the code modules for your classes:

```
Imports System.Reflection
<Assembly: AssemblyVersionAttribute("2.3.4.6")>
```

These lines have to go outside the classes you have created.

Self Learning Exercise-6

i. Fill in the Blanks

k) Version numbers have four components: major version, minor version, _________ and ______________.

ii. State True/False

i) An application can contains only one assembly.

8.8 Attributes

Attributes are declarative tags that can be used to annotate types or class members, thereby modifying their meaning or customizing their behavior. This descriptive information provided by the attribute is stored as metadata in a .NET assembly and can be extracted either at design time or at runtime using reflection. To see how attributes might be used, consider the <WebMethod> attribute, which might appear in code as follows:

```
<WebMethod(Description:="Indicates the number of visitors to a page")> _
        Public Function PageHitCount(strULR As String) As Integer
```

Ordinarily, public methods of a class can be invoked locally from an instance of that class; they are not treated as members of a web service. In contrast, the <WebMethod> attribute marks a method as a function callable over the Internet as part of a web service. This <WebMethod> attribute also includes a single property, Description, which provides the text that will appear in the page describing the web service.

You may feel why attributes are used on the .NET platform and why they are not simply implemented as language elements. The answer comes from the fact that attributes are stored as metadata in an assembly, rather than as part of its executable code. As an item of metadata, the attribute describes the program element to which it applies and is available through reflection both at design time (if a graphical environment such as Visual Studio .NET is used), at compile time (when the compiler can use it to modify, customize, or extend the compiler's basic operation), and at runtime (when it can be used by the Common Language Runtime or by other executable code to modify the code's ordinary runtime behavior).

8.8.1 Syntax and Use

In Visual Basic, an attribute appears within angle brackets (a less-than (<) and a greater-than symbol (>)). The attribute name is followed by parentheses, which are used to enclose arguments that might be passed to the attribute. For example, the <Obsolete> attribute marks a type or type member as obsolete. We can apply <Obsolete> as a parameter-less attribute as follows:

<Obsolete( )>

If no arguments are assigned to the attribute, we can omit the trailing parentheses:

<Obsolete>

If more than one attribute is applied to a single program element, the attributes are enclosed in a single set of angle brackets and delimited from one another by a comma. For example:

<Obsolete(), WebMethod( )> Public Function PageCount(strURL As String) As Integer

Each attribute corresponds to a class derived from System.Attribute. The VB.NET compiler treats an attribute as an instance of the attribute's class.) By convention, we drop the trailing string "Attribute" from the class name to form the attribute name, although the attribute name can also be identical to the class name. Thus, for example, the <WebMethod> attribute corresponds to the WebMethodAttribute class in the System.Web.Services namespace, which in turn is found in *System.Web.Services.dll*. Alternately, you can also specify the attribute as <WebMethodAttribute>. If the namespace containing the attribute class is not automatically accessible to the Visual Basic compiler or to Visual Studio, the Imports directive should be used, and a reference should be added to the project either using the References dialog in Visual Studio or the /r switch in the command-line compiler. Required arguments must be supplied to the attribute as positional arguments *only*; named arguments are not accepted. A comma separates all arguments, whether named or positional.

Unless it has a modifier, an attribute immediately precedes the language element to which it applies and must be on the same logical line as that language element. If they are on different lines, the Visual Basic .NET line continuation character (the underscore, or _) must be used. This syntax is valid for attributes applied to the following language elements:

- Class
- Constructor
- Delegate
- Enum
- Event
- Field
- Interface
- Method
- Parameter
- Property
- Return Value
- Structure

For example, the following Class statement illustrates this general usage of an attribute:

```
<AttributeUsage(AttributeTargets.All)> Public Class MyCustomAttrAttribute
```

The following statement indicates how attributes are used with parameter declarations:

```
Public Sub FunctionOne(strName As String, _
          <ParamArrayAttribute( )> lValues As Long)
```

There are two exceptions to this rule. Some attributes must be prefaced with a modifier (either Assembly: or Module:) indicating the program element to which the attribute applies. In that case, the attribute must be placed at the top of the source file (i.e., immediately following any Option and Imports statements), along with any other attributes that require a modifier. This syntax is valid for an attribute applied to an assembly or a module only.

Self Learning Exercise-7

i. Fill in the Blanks

l) _________are declarative tags that can be used to annotate types or class members

m) Each attribute corresponds to a class derived from ____________

ii. State True/False

j) In Visual Basic, an attribute appears within angle brackets (a less-than (<) and a greater-than symbol (>)).

8.9 Summary

In this unit we have discussed what is IDE, the features and components of IDE. We have discussed the process of migration from VB to VB.NET, the concept of Namespace, Attribute and Versioning and types of namespace used in VB.NET.

8.10 Glossary

.NET FrameWork	Type-safe environment for developing and executing applications.
Attribute	Is a declarative tag that can be used to annotate types or class members
Class Library	Provides functionality similar to ActiveX and DLL by creating classes that access other applications.
CLR	Provides the environment within which the programs run.
COM Interoperability	COM interoperability enables you to leverage your existing code assets and offers seamless bi-directional communication between Visual Basic 6.0 and Visual Basic .NET applications.
Console Application	Command line based applications.
IDE	Is the Development Environment for all .NET based applications which comes with rich features
Namespace	Is a collection of different classes, can be seen as a container for some classes
Version Number	The .NET version number consist four parts:<major number>,<minor number>,<build>,<revision>
Windows Application	allows to create standard windows based applications

Windows Control Library	allows to create our own windows controls
XML Web Service	Enable you to call components running on any platform using open Internet protocols.

8.11 Further Readings AND REFERENCES

1 Gary Cornell, Jonathan Morrison, "Programming in VB.NET: a guide for experienced Programmers", Apress publication, Delhi

2 Steven Holzner ,"Visual Basic .NET Programming: Black Book", Dreamtech Press, New Delhi.

3 Tony Gaddis, Kip Irvine, Bruce Quenton ,"Starting out with Visual Basic .NET programming", Dreamtech Press, New Delhi.

4 Jeffery R Shapiro, "Visual Basic .NET: The Complete Reference", Tata Mcgraw Hill, New Delhi

5 www.startvbdotnet.com

8.12 Answers to Self Learning Exercises

i. Fill in the Blanks	ii.True/False
a) Web Based	a) True
b) Console Applications	b) False
c) Class Libraries	c) True
d) Cross-language compatibility	d) True
e) Rapid Application Development	e) True
f) XML web services	f) True
g) Horizontal migration	g) False
h) Horizontal migration	h) True
i) System	i) False
j) Global Namespaces	j) True
k) build number, revision number	
l) *Attributes*	
m) System.Attribute	

8.13 Unit End Questions

1. What is Cross Language compatibility?
2. Explain the two types of migration Strategies?
3. Which are the two types of namespaces?
4. How can we set the version number?
5. Is it possible to omit the trailing parenthesis of a attribute? If yes, when?
6. Explain the features of the VB.NET.
7. Explain the .NET framework.

8. What is Common Language Runtime? Explain it’s working?
9. Explain the methods of migrating from VB to VB.NET.
10. Explain at least seven languages supported by .NET framework.

Unit - 9: Creating Applications in VB.NET

Structure of the Unit

9.0 Objectives

After going through this unit you will be able to

- Use Various control structures and
- Know about Data types available in VB.NET
- Understand windows form
- Use various controls available with form
- Create standalone applications in .NET environment i.e. in VB.NET.

9.1 Introduction

From the previous Unit you have learned about .NET framework you have also learned what is IDE i.e. Integrated Development Environment.

In this Unit we will discuss about Windows forms and various controls that can be placed on the form.

Before going through this unit you are supposed to have an overview of Unit-8 and Unit-2.

9.2 Data Types and Conversion Functions

All of the languages under the .NET umbrella now implement a subset of a common set of data types, defined in the .NET Framework's Base Class Library (BCL). Because all .NET languages share a consistent and structured way to represent data, bugs and security holes due to the way information is stored in a program are a thing of the past.

9.2.1 Data Types

Boolean	2 bytes
Byte	1 byte
Char	2 bytes
Date	8 bytes
Decimal	16 bytes

Table 9.1: Data types in VB.Net

Type	Storage Size	Value Range
Boolean	2 bytes	True or False
Byte	1 byte	0 to 255 (unsigned)
Char	2 bytes	0 to 65535 (unsigned)
Date	8 bytes	January 1, 0001 to December 31, 9999
Decimal	16 bytes	+/- 79,228,162,514,264,337,593,543 ,950,335 with no decimal point+/- 7.9228162514264337593543950335 with 28 places to right of the decimal point, smallest non-zero number is +/- 0.0000000000000000000000000001
Double	8 bytes	1.79769313486231E+308 to 4.94065645841247E-324 for positive values, 4.94065645841247E-324 to 1.79769313486231E+308 for negative values

Integer	4 bytes	-2147483648 to 2147483647
Long	8 bytes	9,223,372,036,854775808 to 9,223,372,036,854775807
Object	4 bytes	Any type can be stored in a variable of type object
Short	2 bytes	-32768 to 32767
Single	4 bytes	3.402823E+38 to 1.401298E-45 for negative values and 1.401298E-45 to 3.402823E+38 for positive values
String	Depends on implementation platform	0 to approximately 2 billion Unicode characters
User defined type(structure)	Sum of sizes of its members, each member of the structure has a rang determined by its data type and dependent of the ranges of other members	

9.2.2 Conversion Functions

Table 9.2: Conversion Functions

Cbool	Converts to Bool data type
Cbyte	Converts to Byte data type
Cchar	Converts to Character data type
Cdate	Converts to Date data type
CDbl	Converts to Double data type
Cdec	Converts to Decimal data type
Cint	Converts to Integer data type
CLng	Converts to Long data type
Cobj	Converts to Object type
Cshort	Converts to Short data type
CSng	Converts to Single data type
CStr	Converts to String data type

Conversion between character and character codes

ASC: Takes a character and returns its character code e.g. ASC(“A”) returns 65

Chr: Takes a character code and returns its corresponding character e.g. Chr(65) returns A

9.3 Operators and their Precedence

An operator is a symbol (e.g., =, +, >, etc.) that causes VB.NET to take an action. That action might be an assignment of a value to a variable, the addition of two values, or a comparison of two values, etc.

The rules of precedence tell the compiler which operators to evaluate first. As is the case in algebra, multiplication has higher precedence than addition, so 5 + 7 * 3 is equal to 26 rather than 36. Both addition and multiplication have higher precedence than assignment, so the compiler will do the math, and then assign the result (26) to myVariable only after the math is completed. In VB.NET, parentheses are used to change the order of precedence much as they are in algebra.

9.3.1 Operator Type

There are four major types of operators in VB.Net, Assignment, Mathematical, Relational and Logical.

The assignment operator causes the operand on the left side of the operator to have its value changed to whatever is on the right side of the operator. The following expression assigns the value 15 to myVariable:

Dim myVariable As Integer = 15

VB.NET uses seven mathematical operators: five for standard calculations (+, -, *, /, and \), a sixth to return the remainder when dividing integers (Mod), and a seventh for exponential operations (^).

Relational operators are used to compare two values and then return a Boolean (i.e., true or false). The greater-than operator (>), for example, returns true if the value on the left of the operator is greater than the value on the right. Thus, 5>2 returns the value true, while 2>5 returns the value false.

If statements test whether a condition is true. Often you will want to test whether two conditions are both true, or only one is true, or neither is true. VB.NET provides a set of logical operators for this.

Table 9.3: Operators and their Functions

Arithmetic (Mathematical)	**Assignment**
^ Exponentiation	= Assignment
* Multiplication	^=Exponentiation followed by Assignment
/	/= Division followed by Assignment
\	\= Integer division followed by Assignment
Mod	+= Addition followed by Assignment
+	-= Subtraction followed by Assignment
-	&= Concatenation followed by Assignment
Comparison (Relational)	**Concatenation**
< Less than	&
> Greater than	+
<= Less than or equal to	**Logical/Bitwise**
>= Greater than or equal to	NOT reverse the logical value of its operand
<> Not equal to	OR performs the logical or operation
IS True if two objects refer to same address	XOR performs the logical xor operation (true if either operand is true, false otherwise)
LIKE performs string pattern matching	AND performs the logical and operation (true if both operands are true, false otherwise)

9.3.2 Precedence of Operators

The rules of precedence tell the compiler which operators to evaluate first. As is the case in algebra, multiplication has higher precedence than addition, so 5+7*3 is equal to 26 rather than 36. Both addition and multiplication have higher precedence than assignment, so the compiler will do the math, and then assign the result (26) to myVariable only after the math is completed.

In VB.NET, parentheses are also used to change the order of precedence much as they are in algebra.

The following is the precedence of operators.

^, - ,*/ ,\ ,mod, +- ,+ ,& ,= ,<>, < , >, >=,<=, like, IS, NOT, ANS, OR, XOR

9.4 Conditional Constructs and Loops

While methods branch unconditionally, often we want to branch within a method depending on a condition that we evaluate while the program is running. This is known as conditional branching. Conditional branching statements allow us to write logic such as "If you are over 25 years old, then you may rent a car."

VB.NET provides a number of constructs that allow us to write conditional branches into our programs; these constructs are described in the following sections.

9.4.1 If…Else…End if

If conditional expression is one of the most useful control structures which allows us to execute an expression if a condition is true and execute a different expression if it is False. The syntax looks like this:

If condition **Then**

[statements]

Else If condition **Then**

[statements]

-

-

Else

[statements]

End If

9.4.2 Select…Case

The Select Case statement executes one of several groups of statements depending on the value of an expression. If our code has the capability to handle different values of a particular variable then we can use a Select Case statement. We use Select Case to test an expression, determine which of the given cases it matches and execute the code in that matched case.

The syntax of the Select Case statement looks like this:

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

9.4.3 Do Loop

The Do...Loop allows us to run one or more lines of code repetitively, but we don't know how many times. The Do...Loop is generally slower than the For...Loop, as it tests the value of a expression in each loop.

Do { **While** | **Until** } condition

```
  [ statements ]
[ Exit Do ]
  [ statements ]
Loop
-OR-
Do
  [ statements ]
[ Exit Do ]
  [ statements ]
Loop { While | Until } condition
```

Parts

While

Required unless **Until** is used. Keyword. Repeat the loop until *condition* is **False**.

Until

Required unless **While** is used. Keyword. Repeat the loop until condition is **True**.

condition

Optional. **Boolean**. Expression that evaluates to a value of **True** or **False**.

statements

Optional. One or more statements that are repeated while, or until, condition is **True**.

Table 9.4: The 4 different types of Do loops

Type of Do Loop	**Explanation**	**Example**
Do While … Loop	The Do While ... Loop evaluates the condition, and if the condition is true, then it evaluates the statements following the condition. When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the statements again. It continues repeating this process until the condition is false.	Do While condition Statements Loop
Do Until ... Loop	The Do Until ... Loop is similar to the Do While ... Loop except it keeps evaluating the statements until the condition is true rather than while it is true.	Do Until condition Statements Loop
Do ... Loop While	The Do ... Loop While evaluates the statements only once. It then evaluates the condition, and if the	Do statements Loop While

	condition is true, evaluates the statements again. This process continues until the condition is false.	condition
Do ... Loop Until	Similar to Do ... Loop While except that it evaluates the statements until the condition is true.	Do statements Loop Until condition

Note The Do loops support an Exit Do statement to immediately jump out of the loop. The Exit Do statement is similar to the Exit For in For/Next loops.

9.4.4 While Loop

A While loop can be used to execute a fixed block of statement an indefinite amount of time.

Table 9.5: Two Types of While Loops

Type of Loop	**Explanation**	**Example**
While ... Do	The While ... Do loop evaluates the condition, and if the condition is true, then it evaluates the expression following the Do. When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the expression following the Do again. It continues repeating this process until the condition is false.	While condition Do expression
Do ... While	The Do ... While loop evaluates the expression once no matter what. It then evaluates the condition, and if the condition is true, evaluates the expression again. This process continues until the condition is false.	Do expressionWhile condition

The While loops support an Exit While statement to immediately jump out of the loop. Its use is analogous to the use of Exit For in For loops. As with the For loop, the While loop when considered as an expression always returns the Boolean value True.

The While loop is similar to the Do While ... Loop except that it does not support an Exit statement. It uses While ... Wend instead of Do While ... Loop as its syntax.

```
While condition
  statements
Wend
```

9.4.5 For…Next Loop

```
For Each element In group
  [ statements ]
```

[**Exit For**]

  [statements]

Next [element]

Parts

element

Required. Variable. Used to iterate through the elements of the collection or array. The data type of element must be such that the data type of the elements of group can be implicitly converted to it.

group

Required. Object variable. Must refer to an object collection or array.

statements

Optional. One or more statements between **For Each** and **Next** that are executed on each item in group.

The **For Each...Next** loop is entered if there is at least one element in *group*. Once the loop has been entered, the statements are executed for the first element in *group*; if there are more elements in *group*, the statements in the loop continue to execute for each element. When there are no more elements, the loop is terminated and execution continues with the statement following the **Next** statement.

Repeats a group of statements a specified number of times.

For counter = start **To** end [**Step** step]

  [statements]

[**Exit For**]

  [statements]

Next [counter]

Parts

counter

Required. Variable. The type of counter is usually **Integer** but can be any elementary numeric type that supports the greater than (>), less than (<), and addition (+) operators.

start

Required. Expression. The initial value of counter. The start expression usually evaluates to type **Integer** but can evaluate to any data type that widens to the type of counter.

end

Required. Expression. The final value of counter. The end expression usually evaluates to type **Integer** but can evaluate to any data type that widens to the type of counter.

step

Optional. Expression. The amount by which counter is incremented each time through the loop. The step expression usually evaluates to type **Integer** but can evaluate to any data type that widens to the type of counter. If not specified, step defaults to 1.

statements

Optional. One or more statements between **For** and **Next** that are executed the specified number of times.

The step argument can be either positive or negative. The value of the step argument determines loop processing as follows:

Step value	**Loop executes if**
Positive or zero	counter <= end
Negative	counter >= end

The expressions start, end, and step are all evaluated only once, when the **For** statement is first encountered. They are not evaluated again, even if the loop statements change their constituent parts.

The counter variable is compared to end every time before the loop is entered. This includes the first time the **For** statement is executed. Therefore, if the value of start is past the value of end when the loop is entered, the loop is not executed, and execution passes immediately to the statement following the **Next** statement.

After the loop statements have executed, step is added to counter. At this point, the **For** statement again compares counter to end. As a result of this comparison, either the statements in the loop execute again, or the loop is terminated and execution continues with the statement following the **Next** statement.

Changing the value of counter while inside a loop can make it more difficult to read and debug your code.

For/Next loops enable you to evaluate a sequence of statements multiple times. This is unlike the If and Select statements where the program passes through each statement at most once during the formula's evaluation.

For/Next loops are best when you know the number of times that the statements needs to be evaluated in advance.

Exit

Exits a procedure or block and transfers control immediately to the statement following the procedure call or the block definition.

Exit { **Do** | **For** | **Function** | **Property** | **Select** | **Sub** | **Try** | **While** }

Parts

Do

Immediately exits the **Do** loop in which it appears. Execution continues with the statement following the **Loop** statement. **Exit Do** can be used only inside a **Do** loop. When used within nested **Do** loops, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.

For

Immediately exits the **For** loop in which it appears. Execution continues with the statement following the **Next** statement. **Exit For** can be used only inside a **For...Next** or **For Each...Next** loop. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where **Exit For** occurs.

Function

Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function** procedure. **Exit Function** can be used only inside a **Function** procedure.

Property

Immediately exits the **Property** procedure in which it appears. Execution continues with the statement that called the **Property** procedure, that is, with the statement requesting or setting the property's value. **Exit Property** can be used only inside a **Property** procedure.

Select

Immediately exits the **Select Case** in which it appears. Execution continues with the statement following the **End Select** statement. **Exit Select** can be used only inside a **Select Case** statement.

Sub

Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub** procedure. **Exit Sub** can be used only inside a **Sub** procedure.

Try

Immediately exits the **Try** or **Catch** block in which it appears. Execution continues with the **Finally** block if there is one, or with the statement following the **End Try** statement otherwise. **Exit Try** can be used only inside a **Try…Catch…Finally** statement.

While

Immediately exits the **While** loop in which it appears. Execution continues with the statement following the **End While** statement. **Exit While** can be used only inside a **While** loop. When used within nested **While** loops, **Exit While** transfers control to the loop that is one nested level above the loop where **Exit While** occurs.

Do not confuse **Exit** statements with **End** statements. **Exit** does not define the end of a statement.

9.5 Arrays

Arrays allow you to refer to a series of variables by the same name and to use a number, called an index or subscript, to tell them apart. This helps you create shorter and simpler code in many situations, because you can set up loops that deal efficiently with any number of elements by using the index number.

Array Dimensions

An array can have one dimension or more than one. The *dimensionality* or *rank* corresponds to the number of subscripts used to identify an individual element. You can specify up to 32 dimensions, although more than three is extremely rare.

Array Size

Every dimension of an array has a nonzero length. The elements of the array are contiguous along each dimension from subscript 0 through the highest subscript of that dimension. Because Visual Basic allocates space for an array element corresponding to each index number, you should avoid declaring any dimension of an array larger than necessary.

Arrays do not have fixed size in Visual Basic. You can change the size of an array after you have created it. The **ReDim** statement assigns a completely new array object to the specified array variable. Therefore, **ReDim** can change the length of each dimension.

Arrays as Objects

Arrays are objects in Visual Basic .NET, so every array type is an individual reference type. This has the following implications:

- An array variable holds a pointer to the data constituting the elements and the rank and length information.
- When you assign one array variable to another, only the pointer is copied.

- No two array variables are considered to be of the same data type unless they have the same rank and element data type.

Array Class

All arrays inherit from the **Array** class in the **System** namespace, and you can access the methods and properties of **System.Array** on any array. For example, the **Rank** property returns the array's rank, and the **Sort** method sorts its elements.

Array Element Type

An array declaration specifies a data type, and all its elements must be of that type. When the data type is **Object**, the individual elements can contain different kinds of data (objects, strings, numbers, and so on). You can declare an array of any of the fundamental data types, of a structure, or of an object class

You can also declare an array that contains other arrays as its elements. In this case, the constituent arrays must all have the same element data type, which you specify in the declaration. An array of arrays is called a jagged array because the constituent arrays do not have to have the same sizes.

There are several different ways to declare array variables.

- The first way is to use empty parentheses and explicitly specify the type of the array:

```
‘Declare x to be a Global variable
‘of Number Array type
Global x () As Number
‘Initialize x
x = Array (10, 20, 30)
‘Declare y to be a Shared variable
‘of String Range Array type
Shared y () As String Range
‘Initialize y
y = Array (“A” To “C”, “H” To “J”)
```

- The second way is to declare the variable without specifying that it is an array and without giving its type and waiting for the first assignment to the variable to completely specify its type:

```
‘Declare y to be a Local variable
‘but do not specify its type
Dim y
‘The type of y is now set to be a String Array
y = Array (“Sun”, “Mon”, “Tue”, “Wed”, “Th”, “Fri”, “Sat”)
```

- The third way is to declare that the variable is an array but not specify its type fully until the first assignment. Assuming the declaration of y above:

```
‘Declare z to be a Local variable that is an Array
```

```
Local z()
'z is set to Array ("Mon", "Tue") and is a String Array
z = y(2 to 3)
```

- The fourth way is to explicitly specify the size of the array during the declaration. If you use this technique, the array is automatically created and default values are used to fill the array. For example, for a Number Array, each element is initialized to 0 and for a String array each element is initialized to the empty string "". Since this type of declaration actually creates the array, you must specify its type with the As clause so that Crystal Reports knows how much storage space to reserve for the array.

```
Dim a(2) As String
'Assign a value to the first element of the array a
a(1) = "good"
a(2) = "bye"
'The & operator can be used to concatenate strings
'the formula returns the String "goodbye"
formula = a(1) & a(2)
```

Assigning Values to Elements of an Array

You can assign values to elements of an array and also use the values of the elements for other computations.

```
Global x() As String
x = Array ("hello", "bye", "again")
x (2) = "once"
'Now x is Array ("hello", "once", "again")
'The statement below would cause an error if not
'commented out since the array has size 3
'x (4) = "zap"
'The formula returns the String "HELLO"
formula = UCase (x (1))
```

The Redim and Redim Preserve keywords can be used to resize an array, which is useful if you want to add extra information to it. Redim erases the previous contents of the array first before resizing it whereas Redim Preserve preserves the previous contents.

```
Dim x () As Number
Redim x (2) 'Now x is Array (0, 0)
x (2) = 20 'Now x is Array (0, 20)
Redim x (3) 'Now x is Array (0, 0, 0)
x (3) = 30 'Now x is Array (0, 0, 30)
Redim Preserve x (4) 'Now x is Array (0, 0, 30, 0)
formula = "finished"
```

Arrays and For/Next loops

Arrays are commonly used with For/Next Loops. The following example creates and then uses the array Array (10, 20, 30, ..., 100) using a For/Next loop.

```
Dim b (10) As Number
Dim i
For i = 1 To 10
  b(i) = 10 * i
Next i
```

Several variables can be declared in a single statement by separating their declarations with commas.

Self Learning Exercise-1

True/False

a) ASC Conversion function takes a character and returns its character code

b) & is a comparison operator

c) Arrays allow you to refer a series of variables by the same name and to use a number.

d) The Dimensionality or rank corresponds to the number of subscripts used to identify an individual element.

e) Arrays have fixed size in Visual Basic.

f) All arrays inherit from Array class in the System namespace

9.6 Windows Forms

In Visual Basic its these Forms with which we work. They are the base on which we build, develop all our user interface and they come with a rich set of classes. Forms allow us to work visually with controls and other items from the toolbox. In VB .NET forms are based on the System. Windows.Forms namespace and the form class is System.Windows.Forms.Form. The form class is based on the Control class which allows it to share many properties and methods with other controls.

Once you click OK a new Form opens with the title, Form1, towards the top-left side of the form and maximize, minimize and close buttons towards the top right of the form. The whole form is surrounded with a border. The main area of the form in which we work is called the Client Area. It’s in this client area we design the user interface leaving all the code to the code behind file. Forms also support events which let’s the form know that something happened with the form, for example, when we double-click on the form, the Form load event occurs. VB .NET also supports forms to be inherited.

Properties

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

Table 9.6: Properties of Windows Form

Appearance Properties	**Description**
BackColor	Gets/Sets the background color for the form
BackgroundImage	Get/Sets the background image in the form
Cursor	Gets/Sets the cursor to be displayed when the user moves the mouse over the form
Font	Gets/Sets the font for the form
ForeColor	Gets/Sets the foreground color of the form
FormBorderStyle	Gets/Sets the border style of the form
RightToLeft	Gets/Sets the value indicating if the alignment of the control's elements is reversed to support right-to-left fonts
Text	Gets/Sets the text associated with this form
Behavior Properties	Description
AllowDrop	Indicates if the form can accept data that the user drags and drops into it
ContextMenu	Gets/Sets the shortcut menu for the form
Enabled	Gets/Sets a value indicating if the form is enabled
ImeMode	Gets/Sets the state of an Input Method Editor
Data Properties	Description
DataBindings	Gets the data bindings for a control
Tag	Gets/Sets an object that contains data about a control
Design Properties	Description
Name	Gets/Sets name for the form
DrawGrid	Indicates whether or not to draw the positioning grid
GridSize	Determines the size of the positioning grid
Locked	Gets/Sets whether the form is locked
SnapToGrid	Indicates if the controls should snap to the positioning grid
Layout Properties	Description
AutoScale	Indicates if the form adjusts its size to fit the height of the font used on the form and scales its controls
AutoScroll	Indicates if the form implements autoscrolling
AutoScrollMargin	The margin around controls during auto scroll
AutoScrollMinSize	The minimum logical size for the auto scroll region
DockPadding	Determines the size of the border for docked controls
Location	Gets/Sets the co-ordinates of the upper-left corner of the form
MaximumSize	The maximum size the form can be resized to
MinimumSize	The minimum size the form can be resized to

Size	Gets/Sets size of the form in pixels
StartPosition	Gets/Sets the starting position of the form at run time
WindowState	Gets/Sets the form's window state
Misc Properties	Description
AcceptButton	Gets/Sets the button on the form that is pressed when the user uses the enter key
CancelButton	Indicates the button control that is pressed when the user presses the ESC key
KeyPreview	Determines whether keyboard controls on the form are registered with the form
Language	Indicates the current localizable language
Localizable	Determines if localizable code will be generated for this object
Window Style Properties	Description
ControlBox	Gets/Sets a value indicating if a control box is displayed
HelpButton	Determines whether a form has a help button on the caption bar
Icon	Gets/Sets the icon for the form
IsMdiContainer	Gets/Sets a value indicating if the form is a container for MDI child forms
MaximizeBox	Gets/Sets a value indicating if the maximize button is displayed in the caption bar of the form
Menu	Gets/Sets the MainMenu that is displayed in the form
MinimizeBox	Gets/Sets a value indicating if the minimize button is displayed in the caption bar of the form
Opacity	Determines how opaque or transparent the form is
ShowInTaskbar	Gets/Sets a value indicating if the form is displayed in the Windows taskbar
SizeGripStyle	Determines when the size grip will be displayed for the form
TopMost	Gets/Sets a value indicating if the form should be displayed as the topmost form of the application
TransparencyKey	A color which will appear transparent when painted on the form

Option and Imports statements:

Option statement sets the number of options for rest of your code. The possibilities are –

Option Explicit: Set to On or Off- On is default. It requires the declaration of all variables before they are used(this is default)

Option Compare: Set to Binary or Text. This specifies if strings are compared using binary or text comparison operators.

Option Strict: Set to On or Off. Off is default. When you assign a value of one type to a variable of another type. Visual Basic will consider that an error if this statement is on.

Note: use option statement first thing in your code.

Imports

Used to import the namespace so you don't have to qualify items in that namespace by listing the entire namespace when you refer to them.

InputBox Function

It is used to get a string of text from the user. The Syntax is-

Public Function InputBox (Prompt as String[, Title as String [,='"' [, Default Response as String='"' [,XPos as integer= -1 [, Ypos as integer= -1]]]]) as String

Prompt: A String expression displayed as the message in Dialog box.

Title: A String expression displayed in the title bar of Dialog box.

Default Response: A String expression displayed in the Textbox as the default response if no other input is provided.

XPos: The distance in the pixels of the left edge of the dialogbox from the left edge of the screen.

YPos: The distance in the pixels of the upper edge of the dialogbox from the top

edge of the screen.

MsgBox Function

It is used to display the message to the user.

The Syntax is-

Public Function MsgBox (Prompt as Object [, Buttons as MsgBoxStyle = MsgBoxStyle.OKOnly [, Title as Object=Nothing]]) as MsgBoxResultArguments

Prompt: A String expression displayed as the message in Dialog box.

Buttons: The sum of values specifying the number and type of Buttons to display, the icon Style to use, the identity of the default button and the modality of the message box.

Title: A String expression displayed in the title bar of Dialog box.

MsgBox Constants

Constant	Value	Description
OKOnly	0	Displays Ok Button
OKCancel	1	Displays Ok and Cancel Buttons
AbortRetryIgnore	2	Displays Abort, Retry and Ignore Buttons
YesNoCancel	3	Displays Yes,No and Cancel Buttons
YesNo	4	Displays Yes and No Buttons
RetryCancel	5	Displays Retry and Cancel Button
Critical	16	Shows Critical icon
Question	32	Shows Warning query icon
Exclamation	48	Shows Warning Message icon
Information	64	Shows Information Message icon
DefaultButton 1	0	First Button is default

DefaultButton2	256	Second Button is default
DefaultButton3	512	Third Button is default

Self Learning Exercise-2

True/False

g) In VB.NET forms are based on System.Forms.Windows.Form class.

h) MsgBox Function is used to display the message to the user

i) Option Compare specifies if strings are compared using binary or text comparison operators.

9.7 Windows Controls

Control is an object that can be drawn on to the Form to enable or enhance user interaction with the application. Examples of controls are TextBoxes, Buttons, Labels, Radio Buttons, etc. All these Windows Controls are based on the Control class, the base class for all controls.

9.7.1 Label, LinkLabel

Label

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

LinkLabel

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

Properties

ActiveLinkColor	: used to set the color of active link.
LinkColor	: used to set the color of link.
LinkVisited	: used to set the color of visited link.

Events

LinkClicked	: occurs when a link is clicked inside the link label

9.7.2 TextBox

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

Some important properties in the Behavior section of the Properties Window for

TextBoxes.

Properties:

Enabled	:	Default value is True. To disable, set the property to False.
Multiline	:	Setting this property to True makes the TextBox multiline which allows to accept multiple lines of text. Default value is False.
PasswordChar	:	Used to set the password character. The text displayed in the TextBox

		will be the character set by the user. Say, if you enter *, the text that is entered in the TextBox is displayed as *.
ReadOnly	:	Makes this TextBox readonly. It doesn't allow to enter any text.
Visible	:	Default value is True. To hide it set the property to False.
Text	:	sets/gets the current text in the control
TextAlign	:	Allows to align the text from three possible options. The default value is left and you can set the alignment of text to right or center.
Scrollbars	:	Allows to add a scrollbar to a Textbox. Very useful when the TextBox is multiline. You have four options with this property. Options are are None, Horizontal, Vertical and Both. Depending on the size of the TextBox anyone of those can be used

Events

TextChanged	:	when the text of control is changed
Click	:	occurs when the text box is clicked

Methods

AppendText	:	Appends text to current text in textbox
Clear	:	clears all the text from the textbox
Copy	:	copies all the selected text in the text box to the clip board
Cut	:	moves the selected text to the clip board
Paste	:	replaces the selected text in the text box with the contents of clip board
Select	:	selects text in the text box

9.7.3 RichTextBox

RichTextBoxes are similar to TextBoxes but they provide some advanced features over the standard TextBox. RichTextBox allows formatting the text, say adding colors, displaying particular font types and so on. The RichTextBox, like the TextBox is based on the TextBoxBase class which is based on the Control class. These RichTextBoxes came into existence because many word processors these days allow us to save text in a rich text format. With RichTextBoxes we can also create our own word processors. We have two options when accessing text in a RichTextBox, text and rtf (rich text format). Text holds text in normal text and rtf holds text in rich text format

Properties

Multiline	:	Setting this property to True makes the TextBox multiline which allows to accept multiple lines of text. Default value is False.
Readonly	:	Makes this TextBox readonly. It doesn't allow to enter any text.
Visible	:	Default value is True. To hide it set the property to False.
Maxlength	:	Used to set the maximum no of characters the user can type into the richtextbox
Lines	:	Used to set no. of lines of text
Wordwrap	:	Used to indicate the control to wrap the words
Text	:	Used to get/set the current text

SelectedText	:	Used to get/set the selected text
SelectedLength	:	Used to get/set the length of selected text

Methods

AppendText	:	Appends the text to the current text in RTB
Clear	:	Clears all the text from the RTB
Cut	:	Moves the current selection in RTB to the clipoard
Copy	:	Copies the current selection in RTB to the clipboard
CanPaste	:	Determines if you can paste information from the clip board
Find	:	Searches the text within the contents of Rich Text Box
Paste	:	Replaces the selected text in the text box with the contents of clip board
LoadFile	:	Loads the contents of specified file into Rich Text Box
SaveFile	:	Saves the contents of Rich Text Box in a file
SelectAll	:	Selects all the text in RTB

Events

Click	:	Occurs when the RTB is clicked
ReadOnlyChanged	:	Occurs when the readonly property is changed
TextChanged	:	Occurs when the text in RTB is changed

Example

Code for creating bold and italic text in a RichTextBox.

Drag a RichTextBox (RichTextBox1) and a Button (Button1) onto the form. Enter some text in RichTextBox1, say, “We are working with RichTextBoxes”. Paste the following code in the click event of Button1. The following code will search for text we mention in code and sets it to be displayed as Bold or Italic based on what text is searched for.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
RichTextBox1.SelectionStart = RichTextBox1.Find(“are”)
‘using the Find method to find the text “are” and setting it’s
‘return property to SelectionStart which selects the text to format
Dim ifont As New Font(RichTextBox1.Font, FontStyle.Italic)
‘creating a new font object to set the font style
RichTextBox1.SelectionFont = ifont
‘assigning the value selected from the RichTextBox the font style
RichTextBox1.SelectionStart = RichTextBox1.Find(“working”)
Dim bfont As New Font(RichTextBox1.Font, FontStyle.Bold)
RichTextBox1.SelectionFont = bfont
End Sub
```

When you run the above code and click Button1, the text “*are*” is displayed in Italic and the text "**working**" is displayed in Bold font. The image below displays the output

9.7.4 Button

One of the most popular control in Visual Basic is the Button Control (previously Command Control). They are the controls which we click and release to perform some action. Buttons are used mostly for handling events in code, say, for sending data entered in the form to the database and so on. The default event of the Button is the Click event and the Button class is based on the ButtonBase class which is based on the Control class.

Properties

BackColor	:	Sets the background color of the button
BackGroundImage	:	Sets the background image of the button
Font	:	Style of the text that appears on the button
Text	:	Caption of the button
TextAlign	:	Specify where on the button the text should appear
Enabled	:	Specify whether control should be enabled at runtime (true by default)
Visible	:	Specify whether control should be visible at runtime(true by default)
Dock	:	Specify the layout of the control on form

Methods

Performclick	:	Causes a click event for a button

Events

Click	:	Causes when button is clicked
Gotfocus	:	Causes when button receives the focus
Lostfocus	:	Causes when button looses the focus

Self Learning Exercise-3

j) Make a label that will take us to "**www.yahoo.com**"

k) Write an application code for setting the Color of Text with particular color

l) Write an application for Saving Files to RTF

9.7.5 CheckBox

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

Properties

BackgroundImag	:	Used to set a background image for the checkbox.
CheckAlign	:	Used to set the alignment for the CheckBox from a predefined list.
Checked	:	Default value is False, set it to True if you want the CheckBox to be displayed as checked.
CheckState	:	Default value is Unchecked. Set it to True if you want a check to appear. When set to Indeterminate it displays a check in gray background.

FlatStyle	:	Default value is Standard. Select the value from a predefined list to set the style of the checkbox.
ThreeState	:	This property is set to False by default. Set it to True to specify if the Checkbox can allow three check states than two.

Events

CheckedChanged	:	Occurs when the checked property is changed

9.7.6 RadioButton

RadioButtons are similar to CheckBoxes but RadioButtons are displayed as rounded instead of boxed as with a checkbox. Like CheckBoxes, RadioButtons are used to select and deselect options but they allow us to choose from mutually exclusive options. The RadioButton control is based on the ButtonBase class which is based on the Control class. A major difference between CheckBoxes and RadioButtons is, RadioButtons are mostly used together in a group.

Properties

BackgroundImage	:	Used to set a background image for the RadioButton.
CheckAlign	:	Used to set the alignment for the RadioButton from a predefined list.
Checked	:	Default value is False, set it to True if you want the RadioButton to be displayed as checked.
FlatStyle	:	Default value is Standard. Select the value from a predefined list to set the style of the RadioButton.

Events

CheckedChanged	:	Occurs when the value of checked property changes.

9.7.7 ListBox

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

Properties

HorizontalScrollBar	:	Displays a horizontal scroll bar to the listbox. Works when the listbox had Multiple Columns.
Multicolumn	:	Default value is false, set it true if you want the list box to display multiple columns
ScrollAlwaysVisible	:	Default value is set to False. Setting it to True will display both Vertical and Horizontal scrollbar always.
SelectionMode	:	Default value is set to one. Select option None if you do not any item to be selected. Select it to MultiSimple if you want multiple items to be selected. Setting it to MultiExtended allows you to select multiple items with the help of Shift, Control and arrow keys on the keyboard.
Sorted	:	Default value is set to False. Set it to True if you want the items displayed in the ListBox to be sorted by alphabetical order.
Items	:	Allows us to add the items we want to be displayed in the list box

Events

Selected Index Changed	:	Occurs when the selected Index property has changed.

9.7.8 ComboBox

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

Properties

Items	:	Gets a collection of the items in combo box.
SelectedItem	:	Returns/sets item in the ComboBox.
SelectedText	:	Returns the selected text in textbox part of Combo Box.
Sorted	:	Sets the items in sorted order.
DropDownStyle	:	Allows us to set the look of the ComboBox. The default value is set to DropDown which means that the ComboBox displays the Text set by it's Text property in the Textbox and displays it's items in the DropDownListBox below. Setting it to simple makes the ComboBox to be displayed with a TextBox and the list box which doesn't drop down. Setting it to DropDownList makes the ComboBox to make selection only from the drop down list and restricts you from entering any text in the textbox.

Methods

FindString	:	Finds the firstitem in the combo box that begins with the indicated string
GetItemText	:	Get the item of an item
Select	:	Selects the range of text.
FindStringExact	:	Finds the item that matches the indicated string exactly.

Events

DropDown	:	Occurs when the drop down portion of ComboBox is shown.
Selected Index Changed	:	Occurs when the selected Index property has changed.

Self Learning Exercise-4

m) Write an application to check a RadioButton's state

n) Write an application that will show the selected item from ListBox in a TextBox

o) Write an application that will remove all the items from Combo Box

9.7.9 Picture Box

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

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

Properties

BorderStyle	:	Indicate the border style for the picture box
Image	:	Indicate the image that is in the picture box

Events

Resize	:	Occurs when the picture Box is resized
SizeModeChanged	:	Occurs when size mode changes

As the picture box is derived directly from the control class, so we can say that picture box inherits the properties, methods and events from control class such as click event.

Adding Images to PictureBox

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

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

9.7.10 Scrollbars

Anyone familiar with windows knows that scrollbars are those vertical or horizontal controls that display a scroll box or thumb that you can manipulate

There are two types of scrollbars- Horizontal & Vertical

Most controls that typically uses scrollbars come with them built in, such as multilane textboxes, list controls and Combo Box. However you can use the scrollbars if you want to do some custom work such as scrolling the image in picture box or letting the user get visual feedback while setting numeric ranges, as when they are setting red, green and blue values to select a color.

Properties

LargeChange	:	Indicate the value added to or subtracted from to the value property when the scrollbar it self is clicked
Maximum	:	Indicate the upper limit of scrollable range
Minimum	:	Indicate the lower limit of scrollable range
SmallChange	:	Indicate the value added to or subtracted from to the value property when user clicks on arrow button
Value	:	Indicate the value corresponding to the current position of scrollbox

Events

Scroll	:	Occurs when the scroll box is moved
ValueChanged	:	Occurs when the value property has changed either by a scroll event or programmatically

9.7.11 Timer

Timers let you create periodic events, timers are no longer controls but components, and they do not appear in the window at run time. At design time they appear in the component tray underneath the form you have added them to.

You set how often you want timer to generate Tick events by setting the interval property(in millisecond)

Each time a Tick event happens, you can execute code in a handler for this event just as you would for any other event.

Note: In VB6 and before you could set a timer's interval property to 0 to disable the timer, but the minimum possible value for this property is now 1. you now use the enabled property to turn timers on/off. You also can use the new start and stop methods to start and stop the timer.

Property

Enabled	:	Indicate whether the timer is running.
Interval	:	Indicate the time (in milliseconds) between timer Ticks

Methods

Start	:	Starts a timer
Stop	:	Stops a timer

Event

Tick	:	Occurs when the timer interval has elapsed (and timer is enabled)

Self Learning Exercise-5

True/False

p) PictureBoxes are used to display text on them

q) There are two types of scrollbars- Horizontal & Vertical

r) Timers let you create periodic events

9.8 Summary

- Options:sets the number of options for rest of your code
- Label: used to display the text that cannot be edited by the user
- LinkLabel: Based on the label class, supports, web-style hyperlinks to the internet and other windows forms
- TextBox: Box like control in which you can enter text.
- Buttons: The plain control that you simply click and release provides the most popular way of creating and handling an event in your code.
- Listbox: displays a list of items, from which user can select one or more items.
- Timers: lets you to create periodic events.
- Windows form: represents the window that will appear in your application. You place the control on the form it self.

9.9 Glossary

AutoSize property	A Label property that, when set to true, causes the label's size to display all the text in the text property
Boolean	A value that can either be true or false
Button	A rectangular button-shaped control that performs an action when clicked with mouse
Caption property	The property of many controls that determines text displays somewhere on the control
CheckBox Control	A control, which may appear alone or in groups, allow the user to make yes/no or on/off selection.
Chr Function	An intrinsic function that accepts a character code as an argument and returns the character that corresponds to the code
ComboBox	A control that is combination of a text box and list box
Do Until loop	A looping structure that causes one or more statements to repeat until its test expression is true
Do While loop	A looping structure that causes one or more statements to repeat as long as an expression is true
Exit Do	Stops the execution of Do..While or Do..Until loop
Exit For	Stops the execution of For…Next loop
For Each…Next loop	A loop designed specifically to access values fro arrays and array like structure
For..Next loop	A loop designed specifically to initialize, test and increment a counter variable
If….Then	A statement that can cause other statements to execute under certain conditions
If….Then…Else	A statement that will execute one group of statements if a condition is true, or another group of statements if the condition is false
If….Then…ElseIf	A statement that is like a chain of If…Then…Else statements. They perform their test s, one after the other, until one of them found to be true.
Index	A number that identifies a specific element within an array
Label	A control that displays the text that cannot be changed or entered by the user
ListBox	A control that appears as box containing list of items
ListBox control	A control that displays the list of items and also allows the user to select one or more items from the list
PictureBox	A control that displays a graphic image.
RadioButton Control	A control that usually appears in groups and allows the user to select one of the several possible options.
Step value	The value added to the counter variable at the end of each iteration of a For….Next loop
Timer Control	Allows an application to automatically execute code at regular time intervals

9.10 Further Readings AND REFERENCES

1 Gary Cornell, Jonathan Morrison:, "Programming in VB.NET: a guide for experience Programmers", Aspress publication, Delhi

2 Steven Holzner, "Visual Basic .NET Programming: Black Book", Dreamtech Press, New Delhi.

3 Tony Gaddis, Kip Irvine, Bruce Quenton, "Starting out with Visual Basic .NET programming", Dreamtech Press, New Delhi.

4 Jeffery R Shapiro, "Visual Basic .NET: The Complete Reference", Tata Mcgraw Hill, New Delhi.

5 www.startvbdotnet.com.

9.11 Answers to Self Learning Exercises

True/False

a) True

b) False

c) True

d) True

e) False

f) True

g) False

h) True

i) True

j) Make a label that will take us to www.yahoo.com

Drag a LinkLabel (LinkLabel1) onto the form. Write the following code on the click event of this label.

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

k) Write an application code for setting the Color of Text with particular color

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
RichTextBox1.SelectionStart = RichTextBox1.Find("are")
'using the Find method to find the text "are" and setting it's return
'property to SelectionStart which selects the text
```

```
RichTextBox1.SelectionColor = Color.Blue
‘setting the color for the selected text with SelectionColor property
RichTextBox1.SelectionStart = RichTextBox1.Find(“working”)
RichTextBox1.SelectionColor = Color.Yellow
End Sub
```

l) Write an application for Saving Files to RTF

Drag two RichTextBoxes and two Buttons (Save, Load) onto the form. When you enter some text in RichTextBox1 and click on Save button, the text from RichTextBox1 is saved into a rtf (rich text format) file. When you click on Load button the text from the rtf file is displayed into RichTextBox2. The code for that looks like this:

```
Private Sub Save_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Save.Click
RichTextBox1.SaveFile(“hello.rtf”)
‘using SaveFile method to save text in a rich text box to hard disk
End Sub
Private Sub Load_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Load.Click
RichTextBox2.LoadFile(“hello.rtf”)
‘using LoadFile method to read the saved file
End Sub
```

The files which we create using the SaveFile method are saved in the bin directory of the Windows Application

m) Write an application to check a RadioButton’s state

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
If RadioButton1.Checked = True Then
TextBox1.Text = “Selected”
Else
TextBox1.Text = “Not Selected”
End If
End Sub
```

n) Write an application that will show the selected item from ListBox in a TextBox

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

o) Write an application that will remove all the items from Combo box

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

True/False

p) False

q) True

r) True

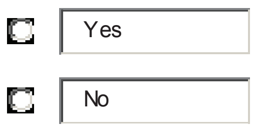
9.12 Unit End Questions

1 Write an application that will to remove a particular item from the List Box.

2 Write an application that will Count the number of Items in ComboBox on the click event of a button

3 Write an application that will display some text when the Checkbox is checked

4 Can you create Windows Services using VB.Net



Yes

No

5 Can you create Command Line applications using VB.Net

Yes

No

6 JIT (in .NET) stands for:

Just in Time

Job in Transit

Java Intermediate Translation

Unit - 10: Object Oriented Programming in VB.NET

Structure of the Unit

10.0 Objectives

After going through this unit you will be able to

- Understand object oriented concepts through VB.Net
- Appreciate Reusability of code
- Implement Inheritance and
- Apply Polymorphism

10.1 Introduction

Up to this unit we have discussed basic controls, their properties, methods and events. We have discussed how to create interface and how to write the code corresponding to interface.

In this unit we will discuss object oriented concepts and their implementation in VB.NET such as implementation of Inheritance, Polymorphism, Encapsulation, Shadowing etc.

You are supposed to take a brief idea of Objects Oriented Concepts before reading this unit.

10.2 Classes and Objects

Visual Basic .NET is Object-Oriented. Everything we do in Visual Basic involves objects in some way or other and everything is based on the Object class. Controls, Forms, Modules, etc. are all types of classes. Visual Basic .NET comes with thousands of built-in classes which are ready to be used. Let's take a closer look at Object-Oriented Programming in Visual Basic. We will see how we can create classes and objects, how to inherit one class from other, what is polymorphism, how to implement interfaces and so on. We will work with Console Applications here as they are simple to code.

Classes are types, and Objects are instances of the Class. Classes and Objects are very much related to each other. Without objects you can't use a class. In Visual Basic we create a class with the Class statement and end it with End Class. The Syntax for a Class looks as follows:

```
Public Class Test
   ——Variables
   ——Methods
   ——Properties
   ——Events
End Class
```

The above syntax creates a class named Test. To create a object for this class we use the new keyword and that looks like this:

```
Dim obj as new Test()
```

or

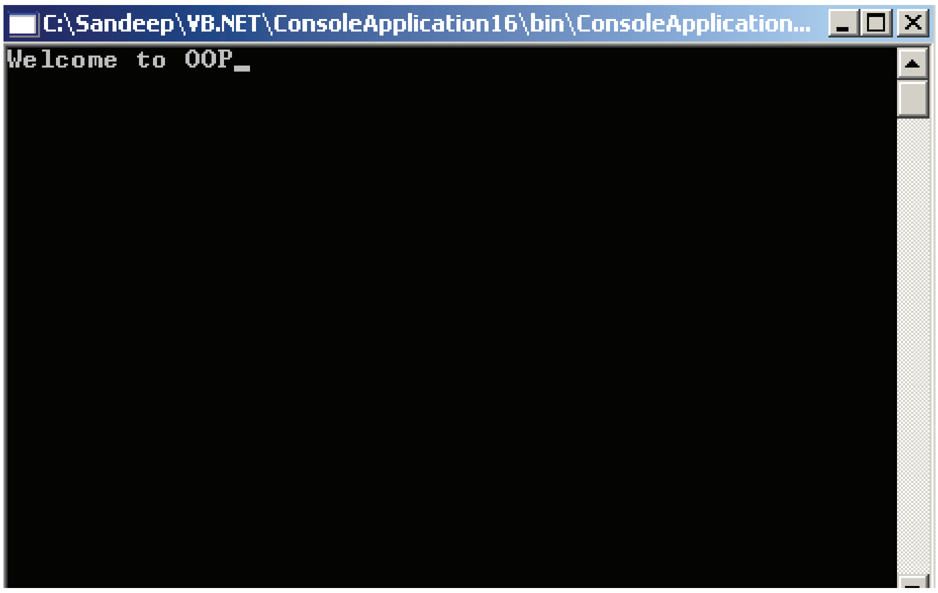
```
Dim object as Test = new Test()
```

The following code shows how to create a Class and access the class with an Object. Open a Console Application and place the following code in it.

```
Module Module1
Imports System.Console
Sub Main()
```

```
Dim obj As New Test()
'creating a object obj for Test class
obj.disp()
'calling the disp method using obj
Read()
End Sub
End Module
Public Class Test
'creating a class named Test
Sub disp()
'a method named disp in the class
Write("Welcome to OOP")
Read()
End Sub
End Class
```

Output of above code is the image below.



```
C:\Sandeep\VB.NET\ConsoleApplication16\bin\ConsoleApplication...
Welcome to OOP_
```

10.2.1 Fields, Properties, Methods and Events

Fields, Properties, Methods, and Events are members of the class. They can be declared as Public, Private, Protected, Friend or Protected Friend.

Fields of a class are also called class's data members.

Fields and Properties represent information that an object contains. Fields of a class are like variables and they can be read or set directly. For example, if you have an object named House, you can store the numbers of rooms in it in a field named Rooms. It looks like this:

```
Public Class House
  Public Rooms As Integer
```

```
End Class
```

Properties are retrieved and set like fields but are implemented using Property Get and Property Set procedures which provide more control on how values are set or returned.

Methods represent the object's built-in procedures. For example, a Class named Country may have methods named Area and Population. You define methods by adding procedures, Sub routines or functions to your class. For example, implementation of the Area and Population methods discussed above might look like this

```
Public Class Country
Public Sub Area()
Write("————")
End Sub
Public Sub population()
        Write("————")
End Sub
End Class
```

Events allow objects to perform actions whenever a specific occurrence takes place. For example when we click a button, a click event occurs and we can handle that event in an event handler.

10.2.2 Creation of Console Based Application

There's another new type of Visual Basic application in VB.NET-console applications. These applications are command line based and run in DOS window. This gives the feeling once again that VB.NET is following lead of Java because Java applications run in DOS window in windows (Visual Basic itself has not interacted with DOS for years), However the change is a welcome one, because it provides us with an option for very simple programming without worrying about user interface implementation and issues.

To create Console application, Select, File->New->Project menu item then select Console application in the template box.

In this case because there is no user interface VB opens the project directly to a code window.

The code looks like this:

```
Module Module1
        Sub main()
        End Sub
End Module
```

Console applications are based on VB Modules that are specifically designed to hold code that is not attached to a form or other such class. Notice the Sub main() and End Sub. When console application is run, the statements inside Sub and End Sub block run automatically.

We can display a message on the console by using WriteLine/Write method, a prewritten method available to us in VB.NET. This method is part of System.Console class, which in turn is part of the System namespace. The System.Console class is part of the .NET framework class library, along with thousands of other classes. To organize all those classes the .NET uses namespace this gives classes their own space and stops conflicts between the various name in such classes

For example:

```
        Module Module1
                Sub main()
                   System.Console.WriteLine("Hello Welcome to VB")
                    Read()
                End Sub
        End Module
```

When you run this code by selecting the Debug->Start menu item, it displays our message and prompt about the enter key like this

```
Hello Welcome to VB
_
```

```
Module Module1
                Sub main()
                        System.Console.Write("Hello Welcome to VB")
                        Read()
                End Sub
End Module
```

When you run this code by selecting the Debug->Start menu item, it displays our message and prompt about the enter key like this

```
Hello Welcome to VB_
```

You can also use write a method so that it displays your message but does not wait for prompt about the enter key.

Fields, properties, methods and events are only part of OOP. Generally speaking a language is object oriented if it supports Abstraction, Encapsulation, Polymorphism and Inheritance.

10.2.3 Constructors and Destructors

A constructor is a special member function whose task is to initialize the objects of it's class. This is the first method that is run when an instance of a type is created. A constructor is invoked whenever an object of it's associated class is created. If a class contains a constructor, then an object created by that class will be initialized automatically. We pass data to the constructor by enclosing it in the parentheses, following the class name when creating an object. Constructors can never return a value, and can be overridden to provide custom initialization functionality. In Visual Basic we create constructors by adding a Sub procedure named New to a class. The following code demonstrates the use of constructors in Visual Basic.

```
Module Module1
        Sub Main()
                Dim con As New Constructor(10)
WriteLine(con.display())
'storing a value in the constructor by passing a value(10) and calling it with the
'display method
```

```
Read()
End Sub
End Module
Public Class Constructor
Public x As Integer
Public Sub New(ByVal value As Integer)
'constructor
x = value
'storing the value of x in constructor
End Sub
Public Function display() As Integer
        Return x
        'returning the stored value
End Function
End Class
```

A destructor, also know as finalizer, is the last method run by a class. Within a destructor we can place code to clean up the object after it is used, which might include decrementing counters or releasing resources. We use Finalize method in Visual Basic for this and the Finalize method is called automatically when the .NET runtime determines that the object is no longer required. When working with destructors we need to use the overrides keyword with Finalize method as we will override the Finalize method built into the Object class. We normally use Finalize method to deallocate resources and inform other objects that the current object is going to be destroyed. Because of the nondeterministic nature of garbage collection, it is very hard to determine when a class's destructor will be called. The following code demonstrates the use of Finalize method.

```
Module Module1
Sub Main()
Dim obj As New Destructor()
End Sub
End Module
Public Class Destructor
        Protected Overrides Sub Finalize()
                Write("hello")
                Read()
        End Sub
End Class
```

When you run the above code, object obj of class, destructor is created and "Hello" is displayed. When you close the DOS window, obj is destroyed.

Self Learning Exercise-1

i. Fill in the Blanks

a) The new keyword is used to create __________ of a class.

b) Class statement ends with ____________.

c) ____________of a class also called its data members.

d) ____________represent the objects built in procedures.

e) In console application ____________ method is used to display message on console.

f) ____________ is a special member function whose task is to initialize the object of its class.

g) A destructor also known as ____________ , is last method run by a class.

10.3 Abstraction

An abstract class is the one that is not used to create objects. An abstract class is designed to act as a base class (to be inherited by other classes). Abstract class is a design concept in program development and provides a base upon which other classes are built. Abstract classes are similar to interfaces. After declaring an abstract class, it cannot be instantiated on it's own, it must be inherited. Like interfaces, abstract classes can specify members that must be implemented in inheriting classes. Unlike interfaces, a class can inherit only one abstract class. Abstract classes can only specify members that should be implemented by all inheriting classes.

Creating Abstract Classes

In Visual Basic .NET we create an abstract class by using the MustInherit keyword. An abstract class like all other classes can implement any number of members. Members of an abstract class can either be Overridable (all the inheriting classes can create their own implementation of the members) or they can have a fixed implementation that will be common to all inheriting members. Abstract classes can also specify abstract members. Like abstract classes, abstract members also provide no details regarding their implementation. Only the member type, access level, required parameters and return type are specified. To declare an abstract member we use the MustOverride keyword. Abstract members should be declared in abstract classes.

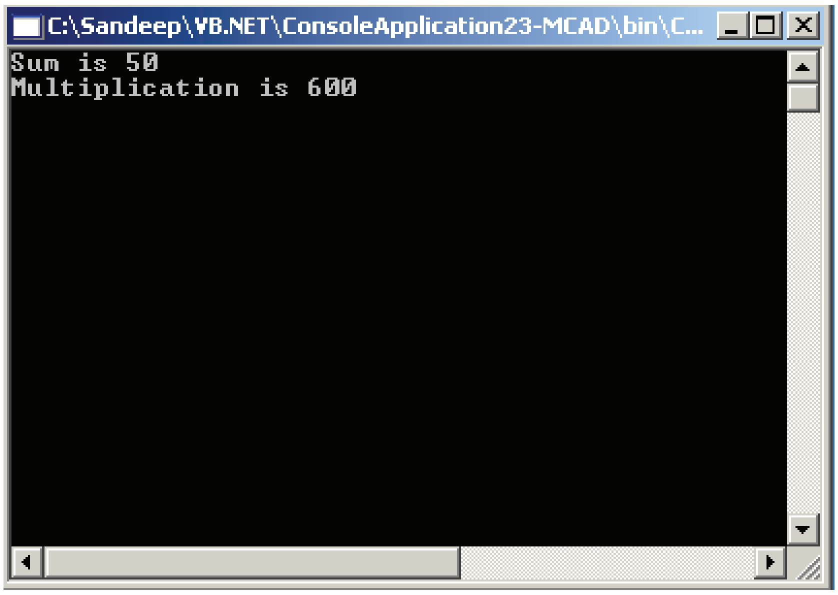
Implementation of Abstract Class

When a class inherits from an abstract class, it must implement every abstract member defined by the abstract class. Implementation is possible by overriding the member specified in the abstract class. The following code demonstrates the declaration and implementation of an abstract class.

```
Module Module1
Public MustInherit Class AbstractClass
‘declaring an abstract class with MustInherit keyword
Public MustOverride Function Add() As Integer
Public MustOverride Function Mul() As Integer
‘declaring two abstract members with MustOverride keyword
End Class
```

```
Public Class AbstractOne
        Inherits AbstractClass
'implementing the abstract class by inheriting
Dim i As Integer = 20
Dim j As Integer = 30
'declaring two integers
                              Public Overrides Function Add() As Integer
Return i + j
End Function
'implementing the add method
Public Overrides Function Mul() As Integer
Return i * j
End Function
'implementing the mul method
End Class
Sub Main()
Dim abs As New AbstractOne()
'creating an instance of AbstractOne
WriteLine("Sum is" & " " & abs.Add())
WriteLine("Multiplication is" & " " & abs.Mul())
'displaying output
Read()
End Sub
End Module
```

The output of above code is the image below.



```
Sum is 50
Multiplication is 600
```

Self Learning Exercise-2

i. Fill in the Blanks

h) We can't instantiate an ____________.

i) To declare an abstract member we use _____________ keyword.

ii. True/False

a) When a class inherits from an abstract class, it must implement every abstract member defined by the abstract class.

10.4 Encapsulation

It is all about the separation between implementation and interface. In other words when you encapsulate an object, you make its code and data internal and no longer accessible to the outside except through a well defined interface. This is also called as data hiding.

Encapsulation allows developers to build objects that can be changed without affecting the client code that uses them. The interface of the object, the set of exposed properties and methods of the object, doesn't change even if the internal implementation does. Encapsulation has been supported by VB since version 4.0.

Implementation of Encapsulation

Lets take a look at two trivial examples of a Person class and implement the same functionality in two different ways.

Person Class Implementation#1

```
Public Class Person
Private m_sFirstName as String
Private m_sLastName as String
Public Property FirstName() as String
Get
    FirstName = m_sFirstName
 End Get
 Set(ByVal Value as String)
    m_sFirstName = Value
 End Set
End Property
Public Property LastName() as String
   Get
        LastName = m_sLastName
   End Get
   Set(ByVal Value as String)
        m_sLastName = Value
   End Set
End Property
```

```
ReadOnly Property FullName() as String
    Get
FullName = m_sLastName & ", " & m_sFirstName
    End Get
 End Property
End Class
```

Person Class Implementation #2

```
Public Class Person
Private m_sFirstName as String
Private m_sLastName as String
Private m_sFullName as String
Public Property FirstName() as String
    Get
FirstName = m_sFirstName
    End Get
   Set(ByVal Value as String)
m_sFirstName = Value
m_sFullName = m_sLastName & ", " & m_sFirstName
    End Set
  End Property
  Public Property LastName() as String
                                          Get
LastName = m_sLastName
    End Get
   Set(ByVal Value as String)
m_sLastName = Value
m_sFullName = m_sLastName & ", " & m_sFirstName
    End Set
  End Property
  ReadOnly Property FullName() as String
     Get
FullName = m_sFirstName
    End Get
  End Property
End Class
```

This illustrates how the internal implementation of the classes is different but the external interface encapsulates how the class works internally. The goal of encapsulation is to allow a developer to transparently use different implementations of the same object.

Self Learning Exercise-3

i. Fill in the Blanks

j) Encapsulation is also known as ____________.

ii. True/False

b) Encapsulation is separation between implementation and interface.

c) When you encapsulate an object, you make only its code internal and no longer accessible to the outside except through a well defined interface .

10.5 Inheritance

A key feature of OOP is reusability. It's always time saving and useful if we can reuse something that already exists rather than trying to create the same thing again and again. Reusing the class that is tested, debugged and used many times can save us time and effort of developing and testing it again. Once a class has been written and tested, it can be used by other programs to suit the program's requirement. This is done by creating a new class from an existing class. The process of deriving a new class from an existing class is called Inheritance. The old class is called the base class and the new class is called derived class. The derived class inherits some or everything of the base class. In Visual Basic we use the Inherits keyword to inherit one class from other. The general form of deriving a new class from an existing class looks as follows:

```
Public Class One
—
—
End Class

Public Class Two
  Inherits One
—
—
End Class
```

Derived classes inherit and can extend the properties, methods, events, fields and constants defined in the base class. The exceptions here are constructors which are not inherited.

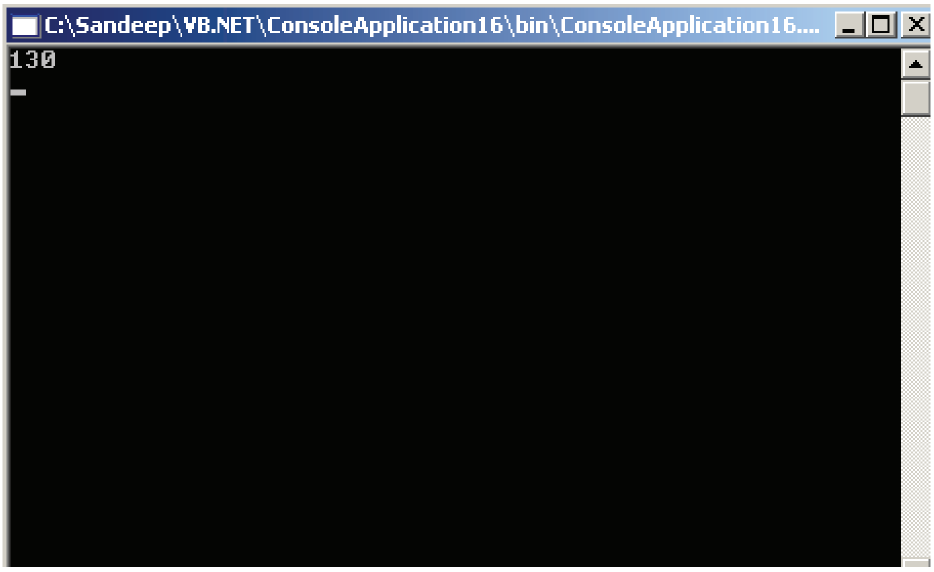
Implementation of Inheritance

Using Inheritance we can use the variables, methods, properties, etc. from the base class and add more functionality to it in the derived class. The following code demonstrates the process of Inheritance in Visual Basic.

```
Imports System.Console
Module Module1
Sub Main()
        Dim ss As New Two()
WriteLine(ss.sum())
Read()
```

```
End Sub
End Module
Public Class One
'base class
            Public i As Integer = 10
Public j As Integer = 20
Public Function add() As Integer
Return i + j
End Function
End Class
Public Class Two
    Inherits One
'derived class. class two inherited from class one
Public k As Integer = 100
Public Function sum() As Integer
'using the variables, function from base class and adding more functionality
            Return i + j + k
End Function
End Class
```

Output of above code is sum of i, j, k as shown in the image below.


C:\Sandeep\VB.NET\ConsoleApplication16\bin\ConsoleApplication16....

```
130
_
```

By default any class can serve as a base class unless you explicitly mark it with NotInheritable keyword

Visual Basic supports multilevel inheritance not multiple inheritance , multiple inheritance is achieved through the feature of multiple interfaces.

10.5.1 Access modifiers

Public: Entities declared as public have public access. There are no restrictions on the accessibility of public entities.

Protected: Entities declared as protected have protected access. They are accessible only from within their own class or from a derived class. You can use protected only at class level.

Friend: Entities declared as friend have friend access. They are accessible only from within the program that contains their declaration and from anywhere else in the same assembly.

Protected Friend: Entities declared as Protected Friend have Protected Friend access. The rules for Protected and Friend apply to Protected Friend as well.

Private: Entities declared as private have Private access. They are accessible only from within their declaration context, including from any nested procedure. You can use private only at module, namespace or file level.

10.5.2 Inheritance Modifiers

By default, all classes can serve as base classes in VB.NET. However you can use two class level modifiers called inheritance modifiers to modify their behavior.

NotInheritable: prevents a class from being used as base class.

MustInherit: Indicates that the class is intended for use as a base class only.

Note: Objects of MustInherit class cannot be created directly, they can be created only as base class instances of derived class.

10.5.3 Use of MyClass and MyBase Keyword

This example will demonstrate you the use of MyClass and MyBase keyword

```
Class Parent
Public Overridable Sub p1()
        Console.Write("Parent.p1")
End Sub
Public Sub p2()
MyClass.p1()
'Implementing keyword MyClass here tells all derived classes to refer to the base / abstract class
wherein the keyword MyClass appears
End Sub
End Class
Class Child
Inherits Parent
Public Overrides Sub p1()
        Console.Write("Child.p1")
MyBase.p2()
End Sub
End Class
```

```
Sub Main()
        Dim p As Parent
Dim c As Child
p = New Parent()
p.p1() 'OK
p.p2() 'OK
p = New Child()
p.p1() 'stack overflow error is prevented
'If MyClass is not implemented above in the base class, you
 will get a stack overflow error.
'So why does the stack oveflow occur?
'*Parent.p2() calls Parent.p1()
'*Parent.p1() is polymorphic because it can be overridden
'*Child.p1() overrides Parent.p1() so any calls to 'Parent.p1() will actually
'call Child.p1() if you have an instance of class Child.
'*Child.p1() calls MyBase.p2()
'*MyBase.p2() is actually Parent.p2()
'Now if you are still with me, the pitfall comes when you have an instance of class Child and 'call
procedures p1 and p2. Calling p1 produces the following execution flow:
'Child.p1 -> Parent.p2 -> Child.p1 -> Parent.p2 -> Child.p1 -> etc.
'and the cycle repeats until we have no stack space left. Calling p2 produces the following
'execution flow:
'Child.p2 -> Child.p1 -> Parent.p2 -> Child.p1 -> Parent.p2 -> Child.p1 -> etc.
'Implementing the keyword MyClass in the base class tells all derived classes to use the method
'in the base class itself and therefore, stack overflow error is prevented
p.p2() 'stack overflow error is prevented
c = New Child()
c.p1() 'stack overflow error is prevented
c.p2() 'stack overflow error is prevented
End Sub
```

Self Learning Exercise-4

i. Fill in the Blanks

k) Reusability can be achieved through ____________.

l) ____________ prevents a class from being used as base class.

ii. True/False

d) The process of deriving a new class from an existing class is called Inheritance.

e) Visual Basic we use the Inherited keyword to inherit one class from other.

10.6 Polymorphism

Polymorphism is one of the vital features of OOP. It means ”one name, many forms”. It is also called Overloading which means the use of same thing for different purposes. Using Polymorphism we can create as many functions we want with one function name but with different argument list. The function performs different operations based on the argument list in the function call. The exact function to be invoked will be determined by checking the type and number of arguments in the function.

VB Handles polymorphism with both late binding and multiple interfaces.

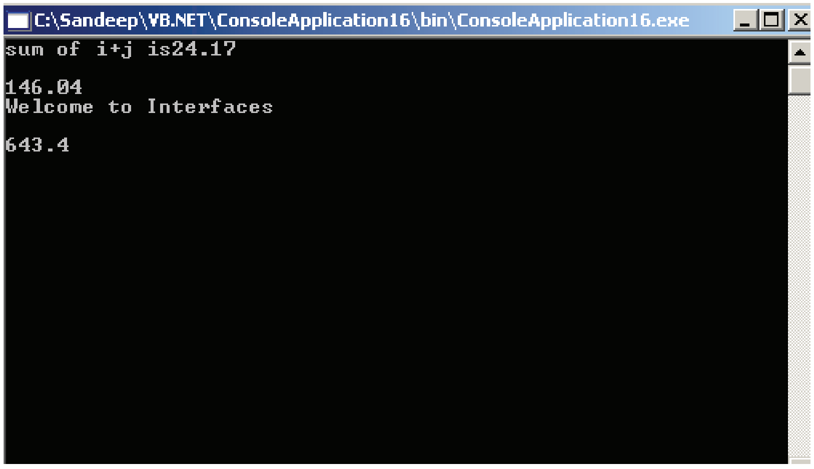
10.6.1 Interfaces

Interfaces allow creation of definitions for component interaction. They also provide another way of implementing polymorphism. Through interfaces, we specify methods that a component must implement without actually specifying how the method is implemented. We just specify the methods in an interface and leave it to the class to implement those methods. Visual Basic .NET does not support multiple inheritance directly but using interfaces we can achieve multiple inheritance. We use the Interface keyword to create an interface and implements keyword to implement the interface. Once you create an interface you need to implement all the methods specified in that interface. The following code demonstrates the use of interface.

```
Imports System.Console
Module Module1
        Sub Main()
                Dim OneObj As New One()
Dim TwoObj As New Two()
‘creating objects of class One and Two
OneObj.disp()
OneObj.multiply()
TwoObj.disp()
TwoObj.multiply()
‘accessing the methods from classes as specified in the interface


End Sub
End Module
Public Interface Test
‘creating an Interface named Test
        Sub disp()
Function Multiply() As Double
‘specifying two methods in an interface
End Interface
Public Class One
  Implements Test
```

```
'implementing interface in class One
        Public i As Double = 12
Public j As Double = 12.17
Sub disp() Implements Test.disp
'implementing the method specified in interface
        WriteLine("sum of i+j is" & i + j)
Read()
End Sub
Public Function multiply() As Double Implements Test.Multiply
'implementing the method specified in interface
WriteLine(i * j)
Read()
End Function
End Class
Public Class Two
  Implements Test
'implementing the interface in class Two
Public a As Double = 20
Public b As Double = 32.17
Sub disp() Implements Test.disp
        WriteLine("Welcome to Interfaces")
        Read()
End Sub
Public Function multiply() As Double Implements Test.Multiply
WriteLine(a * b)
Read()
End Function
End Class
```

Output of above code is the image below.



C:\Sandeep\VB.NET\ConsoleApplication16\bin\ConsoleApplication16.exe

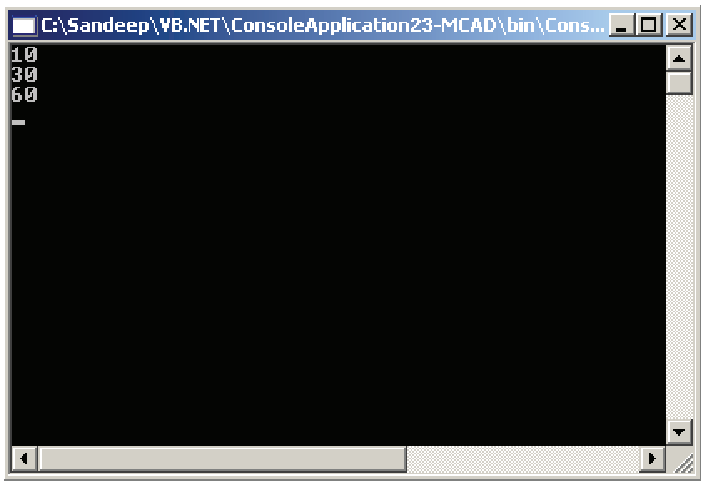
```
sum of i+j is24.17

146.04
Welcome to Interfaces

643.4
```

10.6.2 Function Overloading

The following code demonstrates the implementation of Polymorphism through function overloading.

```
Module Module1
Sub Main()
Dim two As New One()
WriteLine(two.add(10))              ‘calls the function with one argument
WriteLine(two.add(10, 20))          ‘calls the function with two arguments
WriteLine(two.add(10, 20, 30))      ‘calls the function with three arguments
Read()
End Sub
End Module
Public Class One
Public i, j, k As Integer
Public Function add(ByVal i As Integer) As Integer
‘function with one argument
        Return i
End Function
Public Function add(ByVal i As Integer, ByVal j As Integer) As Integer
‘function with two arguments
        Return i + j
End Function
Public Function add(ByVal i As Integer, ByVal j As Integer, ByVal k As Integer) As Integer
‘function with three arguments
Return i + j + k
End Function
End Class
```

Output of the above code is shown in the image below.


C:\Sandeep\VB.NET\ConsoleApplication23-MCAD\bin\Cons...

```
10
30
60
_
```

10.6.3 Function Overriding

By default, a derived class Inherits methods from its base class. If an inherited property or method needs to behave differently in the derived class it can be overridden; that is, you can define a new implementation of the method in the derived class. The Overridable keyword is used to mark a function as overridable. The keyword Overrides is used to mark that a function is overriding some base class function. Let us see an example.

Import the System namespace (already available in .NET).

```
Imports System
```

Our simple base class:

```
Class Human
  ‘Speak() is declared Overridable
  Overridable Public Sub Speak()
    Console.Writeline (“Speaking”)
  End Sub
End Class
```

Now, let us derive a class from Human:

An Indian is a Human:

```
Class Indian
  Inherits Human
  ‘Let us make Indian speak Hindi, the National Language
  ‘in India
  ‘Speak() is overriding Speak() in its base class (Human)
  Overrides Public Sub Speak()
    Console.Writeline (“Speaking Hindi”)
  ‘Important: As you expect, any call to Speak() inside this class
  ‘will invoke the Speak() in this class. If you need to
  ‘call Speak() in base class, you can use MyBase keyword.
  ‘Like this
  ‘Mybase.Speak()
  End Sub
End Class
```

Just a class to put our Main().

```
Class MainClass
  ‘Our main function
  Shared Sub Main()
    ‘Tom is a generic Human
    Dim Tom as Human
    Tom=new Human
    'Tony is a human and an Indian
    Dim Tony as Indian
    Tony=new Indian
    'This call will invoke the Speak() function
    'in class Human
    Tom.Speak()
    'This call will invoke the Speak() function
    'in class Indian
    Tony.Speak()
  End Sub
End Class
```

Self Learning Exercise-5

i. Fill in the Blanks

m) _________________means "one name, multiple forms".

n) ______________ allow us to create definitions for component interaction.

o) Visual Basic .NET does not support multiple inheritance directly but using ______________ we can achieve multiple inheritance.

ii. True/False

f) Polymorphism is known as Data Hiding.

g) Only the early binding is handled by VB.NET.

h) Visual Basic .NET doesn't support multiple inheritance but it can be achieved through feature of multiple interface.

i) Overloading allows us to add new versions of existing methods as long as their parameter lists are different.

j) Method signature of an overridden method can be different from the base class when we use the keyword overrides.

10.7 Shadowing

Shadowing turns out that you can have programming element in the same module, class or structure with the same name but different scope when you have two such elements and the code refers to the name they share, the compiler uses the element with the narrower, closer scope. This is known as shadowing. To implement shadowing we use the keyword shadows.

Overriding as the term says is to override the base class implementation or provide a new implementation in the derived class. We can override members in the base class in VB.Net using the keyword Overrides. Method signature of the overridden methods should be same as the base class methods i.e. overridden members must accept the same data type and number of arguments.

This is a VB.Net concept by which we can provide a new implementation for the base class member without overriding the member. We can shadow a base class member in the derived class by using the keyword "Shadows". The method signature, access level and return type of the shadowed member can be completely different than the base class member.

Overloading allows us to add new versions of existing methods as long as their parameter lists are different. Overriding allows our subclass to entirely replace the implementation of a base class method with a new method that has the same method signature. As we've just seen, we can even combine these concepts to not only replace the implementation of a method from the base class, but also to simultaneously overload that method with other implementations that have different method signatures.

However, any time we override a method using the Overrides keyword, we are subject to the rules governing virtual methods – meaning that the data type of the object controls which implementation of the method is invoked. Sometimes we may want to be able to create a new implementation of a method in our subclass – replacing the implementation in the base class – but we don't want to follow the rules governing virtual methods. In particular, we may want to write client code that can use our object as though it were literally of the base class data type – totally being able to ignore the implementation in the subclass. This is a primary use for the Shadows keyword.

The Shadows keyword can also be used to entirely change the nature of a method or other interface element from the base class – though that is something that should be done with great care since it can seriously reduce the maintainability of our code. Normally, when we create an Employee object, we expect that it can only act as an Employee, but also as a Person since Employee is a subclass of Person. However, with the Shadows keyword we can radically alter the behavior of an Employee class so it doesn't act like a Person. This sort of deviation from what is normally expected invites bugs and makes code hard to understand and maintain. Let's see how Shadows can be used to override non- virtual methods.

Overriding Non-Virtual Methods

Earlier in the chapter we discussed virtual methods and how they are automatically created in VB.NET when the Overrides keyword is employed. We can also implement non-virtual methods in VB.NET. Non-virtual methods are methods that cannot be overridden and replaced by subclasses, and so most methods we implement are non-virtual.

If we don't use the Overridable keyword when declaring a method, it is non-virtual. Non-virtual methods are easy to understand. Since they can't be overridden and replaced, we know that there's only one method by that name, with that method signature and so when we invoke it there is no ambiguity about which specific implementation will be called. The reverse is true with virtual methods, where there may be more than one method of the same name, with the same method signature and so we need to understand the rules governing which implementation will be invoked.

We can override non-virtual methods by using the Shadows keyword. In fact, we can use the Shadows keyword to override methods regardless of whether or not they have the Overridable keyword in the declaration.

This can be very dangerous. The designer of a base class must use care when marking a method as Overridable – ensuring that the base class will continue to operate properly even when that method is replaced by other code in a subclass. Designers of base classes typically just assume that if they don't mark a method as Overridable that it will be called and not overridden. Thus, overriding a non-virtual method by using the Shadows keyword can have unexpected and potentially dangerous side effects since we are doing something that the base class designer assumed would never happen.

Shadowed methods follow different rules than virtual methods when they are invoked. In other words, they don't act like regular overridden methods, but instead they follow a different set of rules to determine which specific implementation of the method will be invoked. In particular, when we call a non-virtual method, it is the data type of the variable that refers to the object that indicates which implementation of the method is called – not the data type of the object as with virtual methods.

To override a non-virtual method we can use the Shadows keyword instead of the Overrides keyword. To see how this works, let's add a new property to our base Person class:

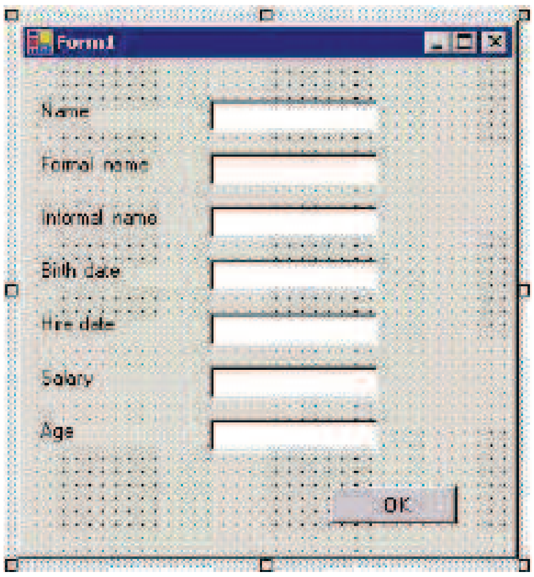
```
Public ReadOnly Property Age() As Integer
    Get
        Return DateDiff(DateInterval.Year, Now(), BirthDate())
    End Get
End Property
```

We've added a new method to our base class – and thus automatically to our subclass – called Age.

This code has a bug – on purpose, for illustration. The DateDiff parameters are in the wrong order, so we'll get negative age values from this routine. Why an intentional bug? Sometimes there are bugs in base classes – sometimes in base classes we didn't write and cannot fix because we don't have the source code. In this case we walk through the use of the Shadows keyword to help address a bug in our base class – acting under the assumption that for some reason we can't actually go fix the code in the Person class. We are not using the Overridable keyword on this method, so any subclass is prevented from overriding the method by using the Overrides keyword. The obvious intent and expectation of this code is that all subclasses will use this implementation and will not override it with their own.

However, the base class cannot prevent a subclass from shadowing a method, and so it doesn't matter whether we use Overridable or not – either way works fine for shadowing.

Before we shadow the method, let's see how it works as a regular non-virtual method. First, we need to change our form to use this new value. Add a text box named txtAge and a related label to the form:



Next let us change the code behind the button to use the Age property. We will also include the code to display the data on the form right here to keep things simple and clear:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnOK.Click
  Dim objPerson As Employee = New Employee()
  With objPerson      .Name = "Fred"
    .BirthDate = #1/1/1960#
        txtName.Text = .Name
        txtBirthDate.Text = Format(.BirthDate, "Short date")
        txtAge.Text = .Age
End With
End Sub
```

Also, change the birth date value to something that will be valid for an Employee, since we don't want that to error out on us now.

At this point we can run the application and the age field should appear in our display as expected – though with a negative value due to the bug we introduced. There is no magic or complexity here – this is basic programming with objects and basic use of inheritance as we discussed at the beginning of this chapter.

Of course we don't want a bug in our code – but if we assume we don't have access to the Person class, and since the Person class doesn't allow us to override the Age method – what are we to do? The answer lies in the Shadows keyword – which allows us to override the method anyway.

Let us shadow the Age method within the Employee class – overriding and replacing the implementation in the Person class even though it is not marked as Overridable. Add the following code to Employee:

```
Public Shadows ReadOnly Property Age() As Integer
        Get
        Return DateDiff(DateInterval.Year, BirthDate(), Now())
        End Get
End Property
```

In many ways this looks very similar to what we have seen with the Overrides keyword, in that we are implementing a method in our subclass with the same name and parameter list as a method in the base class. In this case, however, we'll find some different behavior when we interact with the object in different ways.

Remember that our code in the form is currently declaring a variable of type Employee and is creating an instance of an Employee object:

```
Dim objPerson As Employee = New Employee()
```

This is the simple case, and not surprisingly when we run the application now we will see that the value of the age field is correct – indicating that we just ran the implementation of the Age property from the Employee class. At this point we're seeing the same behavior that we got from overriding with the Overrides keyword.

Let us take a look at the other simple case – where we are working with a variable and object that are both of data type Person. Change the code in Form1 as follows:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles btnOK.Click
   Dim objPerson As Person = New Person()

   With objPerson
      .Name = "Fred"
      .BirthDate = #1/1/1960#
      txtName.Text = .Name
      txtBirthDate.Text = Format(.BirthDate, "Short date")
      txtAge.Text = .Age
   End With
End Sub
```

We have a variable of type Person and an object of that same type. We would expect that the implementation in the Person class would be invoked in this case, and that is exactly what happens – the age field will display the original negative value, indicating that we're invoking the buggy implementation of the method directly from the Person class. Again this is exactly the behavior we would expect from a method overridden via the Overrides keyword.

This next one is where things get truly interesting. Change the code in Form1 as follows:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles btnOK.Click
   Dim objPerson As Person = New Employee()

   With objPerson
      .Name = "Fred"
      .BirthDate = #1/1/1960#

      txtName.Text = .Name
      txtBirthDate.Text = Format(.BirthDate, "Short date")
      txtAge.Text = .Age
   End With
End Sub
```

Now we are declaring the variable to be of type Person, but are creating an object that is of data type Employee. We did this when exploring the Overrides keyword as well – and in that case we discovered that the version of the method that was invoked was based on the data type of the object. The BirthDate implementation in the Employee class was invoked.

If we run the application now we will find that the rules are different when the Shadows keyword is used. In this case, the implementation in the Person class is invoked – giving us the buggy negative value. The implementation in the Employee class is ignored – we get the exact opposite behavior to that with Overrides.

The following table summarizes which method implementation is invoked based on the variable and object data types when using shadowing or non-virtual methods:

Variable	Object	Method invoked
Base	Base	Base
Base	Subclass	Base
Subclass	Subclass	Subclass

In most cases the behavior we'll want for our methods is accomplished by the Overrides keyword and virtual methods. However, in those cases where the base class designer doesn't allow us to override a method and we want to do it anyway, the Shadows keyword provides us with the needed functionality.

Shadowing Arbitrary Elements

The Shadows keyword can be used not only to override non-virtual methods, but it can be used to totally replace and change the nature of a base class interface element. When we override a method we are providing a replacement implementation of that method with the same name and method signature. Using the Shadows keyword we can do more extreme things – like changing a method into an instance variable, or changing a Property into a Function.

However, this can be very dangerous, since any code written to use our objects will naturally assume that we implement all the same interface elements and behaviors as our base class – since that is the nature of inheritance.

By totally changing the nature of an interface element, we can cause a great deal of confusion for programmers who will be interacting with our class in the future.

To see how we can replace an interface element from the base class, let's entirely change the nature of the Age property. In fact, let's change it from being a readonly property method to being a read-write property. We could get even more extreme – changing it to a Function or Sub.

To do this, remove the Age property from the Employee class and add the following code:

```
Public Shadows Property Age() As Integer
  Get
    Return DateDiff(DateInterval.Year, BirthDate(), Now())
  End Get
  Set(ByVal Value As Integer)
    BirthDate() = DateAdd(DateInterval.Year, -Value, Now())
  End Set
End Property
```

With this change, the very nature of the Age method has changed. It is no longer a simple read only property, now it is a read-write property that includes code to calculate an approximate birth date based on the age value supplied.

As it stands, our application will continue to run just fine. This is because we're only using the read- only functionality of the property in our form. We can change the form to make use of the new read- write functionality:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles btnOK.Click
   Dim objPerson As Person = New Person()
   With objPerson
      .Name = "Fred"
      .BirthDate = #1/1/1960#
      .Age = 20
      txtName().Text = .Name
      txtBirthDate().Text = Format(.BirthDate, "Short date")
      txtAge().Text = .Age
   End With
End Sub
```

This will report a syntax error, however.

The variable we are working with, objPerson, is of data type Person – and that data type does not provide a writable version of the Age property. This means that, in order to use our enhanced functionality, we must be using a variable and object of type Employee:

```
Dim objPerson As Employee = New Employee()
```

If we now run the application and click the button we'll see that the Age is displayed as 20, and the birth date is now a value calculated based on that age value – indicating that we are now running the shadowed version of the Age method as implemented in the Employee class.

As if that wasn't odd enough, we can do some even more strange and dangerous things. We can change Age into a variable and we can even change its scope. For instance, we can comment out the Age property code in the Employee class and replace it with the following:

```
Private Shadows Age As String
```

At this point we have changed everything. Age is now a String instead of an Integer. It is a variable instead of a Property or Function. It has Private scope instead of Public scope.

At this point our Employee object is totally incompatible with the Person data type – something that shouldn't occur normally when using inheritance.

This means that the code we wrote in Form1 will no longer work. The Age property is no longer accessible and can no longer be used and so our project will no longer compile. This directly illustrates the danger in shadowing a base class element such that its very nature or scope is changed by the subclass.

Since this change prevents our application from compiling, remove the line in the Employee class that shadows Age as a String variable – and uncomment the shadowed Property routine.

```
Public Shadows Property Age() As Integer
   Get
      Return DateDiff(DateInterval.Year, BirthDate(), Now())
   End Get
```

```
  Set(ByVal Value As Integer)
     BirthDate() = DateAdd(DateInterval.Year, -Value, Now())
  End Set
End Property
```

This will restore our application to a working state and we can move on.

Self Learning Exercise-6

i. Fill in the Blanks

p) To implement shadowing we use the keyword ____________.

q) To override non-virtual method we use the keyword ____________.

r) The shadows keyword can be used not only to override___________, but it can be used to totally replace and change the nature of a base class interface element

ii. True/False

k) When Overridable keyword is not used in the method declaration, it is non-virtual method.

l) If we don't use the Overridable keyword when declaring a method, it is virtual.

10.8 Property Routines and System Windows

You can use both properties and fields to store information in an object.

While fields are simply Public variables, properties use property procedures to control how values are set or returned. You can use the Get and Set keywords for getting/setting properties.

See the following example.

Import the System namespace

```
'Imports System
'Dog is a class.
 Public Class Dog
   'A private variable   to hold the value
   Private mAgeOfDog as Integer
'This is our property routine:
 Public Property Age() As Integer
   'Called when someone tries to retreive the value
Get
   Console.Writeline ("Getting Property")
   Return mAgeOfdog
End Get
Set(ByVal Value As Integer)
   'Called when someone tries to assign a value
   Console.Writeline ("Setting Property")
   mAgeOfDog=Value
```

```
End Set
End Property
End Class
'Another class:
Class MainClass
   'Our main function. Execution starts here.
   Shared Sub Main()
   'Let us create an object.
   Dim Jimmy as Dog
   Jimmy=new Dog
   'We can't access mAgeofDog directly, so we should
   'use Age() property routine.
   'Set it. The Age Set routine will work
   Jimmy.Age=30
   'Get it back. The Age Get routine will work
   Dim curAge=Jimmy.Age()
   End Sub
End Class
```

System Windows: The whole power of Visual Basic has been that you can develop forms, visually, adding controls and other items from the tool box. In VB.NET, the support for windows form is in System.Windows.Forms namespace, and the form class is System.Windows.Forms.Form. The form class itself is based on control class, which means that forms share a lot of properties and methods that controls do.

Self Learning Exercise-7

i. Fill in the Blanks

s) You can use the _____ and _____ keywords for getting/setting properties.

ii. True/False

m) Get/Set keywords are used for Getting/Setting properties.

n) In VB.NET, the support for windows form is in System.Windows.Forms namespace.

10.9 EVENT PROCEDURE

Events in the .NET Framework are based on the delegate model. Delegates are type-safe Function Pointers or Callbacks. A delegate can reference both static and instance methods.

Implementation of an event is a three-step procedure,

(i) Declare a delegate, if definition is not provided the .NET Framework would provide a default delegate implementation.

(ii) Declare the event signature using Event keyword and raise the event using RaiseEvent statement.

(iii) Handle the event by declaring an event receiver, often called event handler, which is a subroutine that responds to the event.

10.9.1 Declare a delegate

Delegate is a class that can hold reference to a method. Delegate class has signature, and it can hold references only to methods that match its signature. Delegate can be declared as

Delegate Sub sampleDel(ByVal Cancel As Boolean)

Any method that has same signature can be attached to this delegate; procedure that would have the same signature can handle this event.

10.9.2 Declare Event and Raise It

Event can be declared in three ways:

evtSample As sampleDel - mechanism to register the event handler for this type of declaration is to be provided by the class declaring the event. The event is implemented by using explicitly declared delegate. The event is raised by making a call to evtSample.

Public Event evtSample as sampleDel - event handler can be registered by using AddHandler method in the Class that would provide the Handler. The event is implemented by using the above-declared delegate. The event is raised by making a call to RaiseEvent.

Public Event evtSample(Cancel as Boolean) - event handler procedure would be registered by using Handles keyword in the declaration itself. The event is implemented by using implicitly declared delegate by the framework. The event is raised by making a call to RaiseEvent.

10.9.3 Handle the Event

Declaring a sub and either attaching it to the delegate or registering with the event declaring class can handle the event.

The example shown declares a delegate in class CTimer and also declares three events in the class using all the aforementioned declarations. The class also provides mechanism to register event and raises events with intervals simulating 5, 10 and 30 units of the interval. Another class CClock is defined that has members to handle the RaisedEvents from CTimer class.

To run the sample from command line use - vbc /out:Event.exe Event.vb

```
Imports System
Public Class CTimer
  Delegate Sub SecondDel(ByVal xintTime As Integer)
  Private evtSecond As SecondDel
  Public Event evtMinute As SecondDel
  Public Event evtHour(ByVal xHour As Integer)
  public Shared lngSeconds As Long
  Public Sub Register(ByVal objSecond As SecondDel)
     evtSecond = evtSecond.Combine(evtSecond, objSecond)
  End Sub
 Public Sub OnTimer()
     lngSeconds = lngSeconds + 1
     If lngSeconds Mod 5 = 0 Then
                              evtSecond(lngSeconds)
```

```
                    End If
    If lngSeconds Mod 10 = 0 Then
      RaiseEvent evtMinute(lngSeconds)
    End If
    If lngSeconds Mod 30 = 0 Then
      RaiseEvent evtHour(lngSeconds)
    End If
  End Sub
End Class
Public Class CClock
  Private WithEvents mobjTimer As CTimer
  Sub New()
    mobjTimer = New CTimer()
    mobjTimer.Register(New CTimer.SecondDel(AddressOf SecondEvent))
    AddHandler mobjTimer.evtMinute, AddressOf MinuteEvent
    While (mobjTimer.lngSeconds < 60)
      mobjTimer.OnTimer()
      System.Threading.Thread.Sleep(100)
    End While
  End Sub
  Private Sub SecondEvent(ByVal xintTime As Integer)
    Console.WriteLine("Second's Event")
  End Sub
  Private Sub MinuteEvent(ByVal xintTime As Integer)
    Console.WriteLine("Minute's Event")
  End Sub
  Private Sub mobjTimer_evtHour(ByVal xintTime As Integer) _
      Handles mobjTimer.evtHour
    Console.WriteLine("Hour's Event")
  End Sub
  Public Shared Sub Main()
    Dim cc1 = New CClock()
  End Sub
End Class
```

Self Learning Exercise-8

i. Fill in the Blanks

t) Events in the .NET framework are based on the ____________

ii. True/False

o) Implementation of an event is three step procedure.

p) Delegate class reference both static and instance methods.

10.10 Summary

- Class: is a type of object also known as blue print.
- Object: is an instance of class, much like a variable is an instance of data type.
- New: keyword is used to instantiate the object of a class.
- Constructor: A special method which is called automatically when you create an object from a class.
- Destructor: Is the last method run by the class, also known as finalizer.
- Encapsulation: A separation between the interface and implementation.
- Inheritance: A reusability feature which allows us to derive a new class from existing one.
- Polymorphism: "one name multiple forms", means the use of same thing for different purposes.
- Interface: allows us to create definition for component interaction.
- Abstraction: ability to create an abstract representation of a concept.
- Shadowing: provides a new implementation for the base class member without overriding the members.
- MustInherit: A keyword that indicates that class is intended for use as a base class
- NotInheritable: A keyword that prevents a class from being used as a base class.

10.11 Glossary

Abstraction	The ability to create an abstract representation of a concept in code
Class	A program structure that defines an abstract data type
Constructor	A class method that is automatically called when an instance of a class is created.
Encapsulation	The hiding of data and procedure inside a class
Finalizer	A class method that is automatically called just before an object is destroyed
Interface	To create definitions for component interaction
MustInherit	Indicates that the class is intended for use as a base class only
NotInheritable	Prevents a class from being used as base class
Object Oriented Programming	A programming technique centered on creating objects. A way of designing and coding applications that had led to using interchangeable software components
Overridable keyword	In a procedure declaration, indicates that the procedure may be overridden in derived class
Polymorphism	One name, multiple forms
Properties	Represent information that an object contains
Shadows	Shadow a base class member in the derived class

10.12 Further Readings AND REFERENCES

1. Gary Cornell, Jonathan Morrison, “Programming in VB.NET: a guide for experience Programmers”, Apress publication, Delhi
2. Steven Holzner, “Visual Basic .NET Programming: Black Book”, Dreamtech Press, New Delhi.
3. Tony Gaddis, Kip Irvine, Bruce Quenton, “Starting out with Visual Basic .NET programming”, Dreamtech Press, New Delhi.
4. Jeffery R Shapiro, “Visual Basic .NET: The Complete Reference”, Tata Mcgraw Hill, New Delhi
5. www.startvbdotnet.com

10.13 Answers to Self Learning Exercise

i. Fill in the Blanks	ii. True/False
a) Instance	a)True
b) End Class	b)True
c) Fields	c)False
d) Methods	d)True
e) Write/WriteLine	e)False
f) Constructor	f)True
g) Finalizer	g)False
h) Abstract Class	h)True
i) MustOverride	i)True
j) Data Hiding	j)False
k) Inheritance	k)True
l) NotInheritable	l)False
m) Polymorphism	m)False
n) Interface	n)True
o) Interface	o)True
p) Shadows	p) True
q) Shadows	q)True
r) non-virtual methods	
s) Get, Set	
t) Delegate Model	

10.14 Unit End Questions

1. The employee list for a company contains employee code, name, designation & basic pay. The employee is given a House Rent Allowance (HRA) of 10% of the basic pay & Dearness Allowance (DA) of 45% of the basic pay. The total pay of the employee is calculated as Basic Pay + HRA + DA. Write a class to define the details of the employee. Write a constructor to assign the required initial values. Add a method to calcu-

late HRA, DA & total pay & print them out. Write another class with main method. Create objects for three different employees and calculate the HRA, DA & total pay.

2. Write a class with a method to find the area of rectangle. Create a subclass to find the volume of a rectangular shaped box.
3. Write a class called Square with a method area(double a) that accepts a argument of type double and return nothing and that finds the area (a^2) of the square. Create a class Cube which is subclass of Square & write an overriding method area(double a) that finds the surface area($6a^2$) of cube.
4. Write an abstract class with a method special that takes an integer as a argument and returns nothing. Create a subclass of this class & implement the special method that finds numbers that are perfect squares from 1 up to n.
5. Write an interface called Numbers, with a method Process that takes two arguments as type of integer and returns an integer value. Write a class called Sum, in which the method Process finds the sum of two numbers & returns an int value. Write another class called Average, in which the Process method finds the average of the two numbers & returns an int.
6. Write an interface called Exam with a method Pass() that accepts an argument of type integer and returns a Boolean value. Write another interface called Classify with a method division (integer average) which returns a String. Write a class called Result which implements both Exam & Classify. The pass method should return true if the mark is greater than or equal to 50 else false. The division method must return “First” when the parameter average is 60 or more, “Second” when average is 50 or more but below 60, “No Division” when average is less than 50
7. What do you mean by Polymorphism? How is the run time binding achieved in VB.NET?
8. Differentiate between

a. Constructor and Destructor

b. Function Overloading and Function Overriding

c. Abstract Class and Interface

9. Write short note on
   a. Encapsulation
   b. Shadowing
   c. Event Procedure

Unit -12: Web and VB.NET

Structure of the Unit

12.0 Objectives

After going through this unit you will be able to

- Get familiar with environment and features of ASP.NET
- Able to learn the concept of client side and server side technologies.

12.1 Introduction

From the previous unit you have learnt about the concepts of .NET framework, namespace, versioning and attribute. In this unit we will discuss what is ASP.NET?, its feature, advantages and concept of web form and controls that can be placed on form. The life cycle of page and the steps for creating and running a web page are also discussed in this unit. Before going through this unit it is required that you must read Unit-8 and solve the exercises.

12.2 Introduction to ASP.NET

ASP.NET, the next version of ASP, is a programming framework used to create enterprise-class Web Applications. These applications are accessible on a global basis leading to efficient information management. The advantages ASP.NET offers is more than just the next version of ASP.

Since 1995, Microsoft has been constantly working to shift it's focus from Windows-based platforms to the Internet. As a result, Microsoft introduced ASP (Active Server Pages) in November 1996. ASP offered the efficiency of ISAPI applications along with a new level of simplicity that made it easy to understand and use. However, ASP script was an interpreted script and consisted unstructured code and was difficult to debug and maintain. As the web consists of many different technologies, software integration for Web development was complicated and required to understand many different technologies. Also, as applications grew bigger in size and became more complex, the number of lines of source code in ASP applications increased dramatically and was hard to maintain. Therefore, an architecture was needed that would allow development of Web applications in a structured and consistent way.

The .NET Framework was introduced with a vision to create globally distributed software with Internet functionality and interoperability. The .NET Framework consists of many class libraries, includes multiple language support and a common execution platform. It's a very flexible foundation on which many different types of top class applications can be developed that do different things. Developing Internet applications with the .NET Framework is very easy. ASP.NET is built into this framework, we can create ASP.NET applications using any of the built-in languages.

Unlike ASP, ASP.NET uses the Common Language Runtime (CLR) provided by the .NET Framework. This CLR manages execution of the code we write. ASP.NET code is a compiled CLR code instead of interpreted code (ASP). CLR also allows objects written in different languages to interact with each other. The CLR makes development of Web applications simple.

12.2.1 Advantages of using ASP.NET

- ASP.NET drastically reduces the amount of code required to build large applications
- ASP.NET makes development simpler and easier to maintain with an event-driven, server-side programming model
- ASP.NET pages are easy to write and maintain because the source code and HTML are together
- The source code is executed on the server. The pages have lots of power and flexibility by this approach

- The source code is compiled the first time the page is requested. Execution is fast as the Web Server compiles the page the first time it is requested. The server saves the compiled version of the page for use next time the page is requested
- The HTML produced by the ASP.NET page is sent back to the browser. The application source code you write is not sent and is not easily stolen
- ASP.NET makes for easy deployment. There is no need to register components because the configuration information is built-in
- The Web server continuously monitors the pages, components and applications running on it. If it noticies memory leaks, infinite loops, other illegal software or activities, it seamlessly kills those activities and restarts itself
- ASP.NET validates information (validation controls) entered by the user without writing a single line of code
- ASP.NET easily works with ADO .NET using data-binding and page formatting features
- ASP.NET applications run faster and counters large volumes of users without performance problems

12.2.2 Differences between ASP.NET and Client-Side Technologies

Client-side refers to the browser and the machine running the browser. Server-side on the other hand refers to a Web server.

Client-Side Scripting

Javascript and VBScript are generally used for Client-side scripting. Client-side scripting executes in the browser after the page is loaded. Using client-side scripting you can add some cool features to your page. Both, HTML and the script are together in the same file and the script is downloaded as part of the page which anyone can view. A client-side script runs only on a browser that supports scripting and specifically the scripting language that is used. Since the script is in the same file as the HTML and as it executes on the machine you use, the page may take longer time to download.

Server-Side Scripting

ASP.NET is purely server-side technology. ASP.NET code executes on the server before it is sent to the browser. The code that is sent back to the browser is pure HTML and not ASP.NET code. Like client-side scripting, ASP.NET code is similar in a way that it allows you to write your code alongside HTML. Unlike client-side scripting, ASP.NET code is executed on the server and not in the browser. The script that you write alongside your HTML is not sent back to the browser and that prevents others from stealing the code you developed.

Self Learning Exercise-1

i. Fill in the Blanks

a) ASP.NET is purely __________technology.

ii. State True or False

a) Javascript and VBScript are generally used for Client-side scripting.

b) Client-side scripting executes in the browser before the page is loaded.

12.3 ASP.NET Features

ASP.NET is not just a simple upgrade or the latest version of ASP. ASP.NET combines unprecedented developer productivity with performance, reliability, and deployment. ASP.NET redesigns the whole process. It's still easy to grasp for new comers but it provides many new ways of managing projects. Below are the features of ASP.NET.

Easy Programming Model

ASP.NET makes building real world Web applications dramatically easier. ASP.NET server controls enable an HTML-like style of declarative programming that let you build great pages with far less code than with classic ASP. Displaying data, validating user input, and uploading files are all amazingly easy. Best of all, ASP.NET pages work in all browsers including Netscape, Opera, AOL, and Internet Explorer.

Flexible Language Options

ASP.NET lets you leverage your current programming language skills. Unlike classic ASP, which supports only interpreted VBScript and JScript, ASP.NET now supports more than 25 .NET languages (built-in support for VB.NET, C#, and JScript.NET), giving you unprecedented flexibility in your choice of language.

Great Tool Support

You can utilize the full power of ASP.NET using any text editor, even Notepad. But Visual Studio .NET adds the productivity of Visual Basic-style development to the Web. Now you can visually design ASP.NET Web Forms using familiar drag-drop-doubleclick techniques, and enjoy full-fledged code support including statement completion and color-coding. VS.NET also provides integrated support for debugging and deploying ASP.NET Web applications. The Enterprise versions of Visual Studio .NET deliver life-cycle features to help organizations plan, analyze, design, build, test, and coordinate teams that develop ASP.NET Web applications. These include UML class modeling, database modeling (conceptual, logical, and physical models), testing tools (functional, performance and scalability), and enterprise frameworks and templates, all available within the integrated Visual Studio .NET environment.

Rich Class Framework

Application features that used to be hard to implement, or required a 3rd-party component, can now be added in just a few lines of code using the .NET Framework. The .NET Framework offers over 4500 classes that encapsulate rich functionality like XML, data access, file upload, regular expressions, image generation, performance monitoring and logging, transactions, message queuing, SMTP mail, and much more. With Improved Performance and Scalability ASP.NET lets you use serve more users with the same hardware.

Compiled execution

ASP.NET is much faster than classic ASP, while preserving the "just hit save" update model of ASP. However, no explicit compile step is required. ASP.NET will automatically detect any changes, dynamically compile the files if needed, and store the compiled results to reuse for subsequent requests. Dynamic compilation ensures that your application is always up to date, and compiled execution makes it fast. Most applications migrated from classic ASP see a 3x to 5x increase in pages served.

Rich output caching

ASP.NET output caching can dramatically improve the performance and scalability of your application. When output caching is enabled on a page, ASP.NET executes the page just once, and saves the result in memory in addition to sending it to the user. When another user requests the same page, ASP.NET serves the cached result from memory without re-executing the page. Output caching is configurable, and can be used to cache individual regions or an entire page. Output caching can dramatically improve the performance of data-driven pages by eliminating the need to query the database on every request.

Web-Farm Session State

ASP.NET session state lets you share session data user-specific state values across all machines in your Web farm. Now a user can hit different servers in the Web farm over multiple requests and still have full access to her session. And since business components created with the .NET Framework are free-threaded, you no longer need to worry about thread affinity.

Enhanced Reliability

ASP.NET ensures that your application is always available to your users.

Memory Leak, DeadLock and Crash Protection

ASP.NET automatically detects and recovers from errors like deadlocks and memory leaks to ensure your application is always available to your users. For example, say that your application has a small memory leak, and that after a week the leak has tied up a significant percentage of your server's virtual memory. ASP.NET will detect this condition, automatically start up another copy of the ASP.NET worker process, and direct all new requests to the new process. Once the old process has finished processing its pending requests, it is gracefully disposed and the leaked memory is released. Automatically, without administrator intervention or any interruption of service, ASP.NET has recovered from the error.

Easy Deployment

ASP.NET takes the pain out of deploying server applications. "No touch" application deployment. ASP.NET dramatically simplifies installation of your application. With ASP.NET, you can deploy an entire application as easily as an HTML page, just copy it to the server. There is no need to run regsvr32 to register any components, and configuration settings are stored in an XML file within the application.

Dynamic update of Running Application

ASP.NET now lets you update compiled components without restarting the web server. In the past with classic COM components, the developer would have to restart the web server each time he deployed an update. With ASP.NET, you simply copy the component over the existing DLL, ASP.NET will automatically detect the change and start using the new code.

Easy Migration Path

You don't have to migrate your existing applications to start using ASP.NET. ASP.NET runs on IIS side-by-side with classic ASP on Windows 2000 and Windows XP platforms. Your existing ASP applications continue to be processed by ASP.DLL, while new ASP.NET pages are processed by the new ASP.NET engine. You can migrate application by application, or single pages. And ASP.NET even lets you continue to use your existing classic COM business components.

XML Web Services

XML Web services allow applications to communicate and share data over the Internet, regardless of operating system or programming language. ASP.NET makes exposing and calling XML Web Services simple. Any class can be converted into an XML Web Service with just a few lines of code, and can be called by any SOAP client. Likewise, ASP.NET makes it incredibly easy to call XML Web Services from your application. No knowledge of networking, XML, or SOAP is required.

Mobile Web Device Support

ASP.NET Mobile Controls let you easily target cell phones, PDAs and over 80 mobile Web devices. You write your application just once, and the mobile controls automatically generate WAP/WML, HTML, or iMode as required by the requesting device.

Self Learning Exercise-2

ii. State True or False

c) ASP.NET automatically detects and recovers from errors like deadlocks and memory leaks to ensure your application is always available to your users.

12.4 ASP.NET Development Environment

Most of the programming languages with which we work require a development environment to code, test and run the programs. You may purchase a copy of that software at your local computer store and work with it. With ASP.NET things are different. ASP.NET is a development technology that is built into the .NET Framework. You can create ASP.NET applications with a simple editor like a notepad. If you prefer to work in a development environment then you have many to choose from. Visual Studio .NET (should purchase) is one development environment from Microsoft. Another development environment from Microsoft which is preferred by many developers is Microsoft Web Matrix. The best thing about ASP.NET Web Matrix is it's free (available as a 1.4 MB free download) and provides most of the features Visual Studio .NET provides.

12.4.1 Setting Up the Development Environment

ASP.NET, you know, is based on the CLR, class libraries and other tools which are integrated into the .NET Framework. To develop and run an ASP.NET application you need to have the .NET Framework installed on your machine. .NET Framework comes pre installed with Operating Systems like Windows 2003 Server and Windows XP. For other operating systems (Windows 2000, 98, Me, NT 4.0) you need to install the .NET Framework manually. You can install .NET Framework manually in two ways: .NET Framework SDK or VS .NET.

Installing the .NET Framework with SDK is simple. Download .NET Framework from Microsoft.com and double-click setup file and follow the instructions. Installing .NET Framework with Visual Studio .NET is simple too. When you install Visual Studio .NET you will be prompted to insert the disk that contains the .NET Framework.

IIS

To develop a Web Application you need IIS (Internet Information Server) on your machine. IIS comes pre installed in Operating Systems like Windows 2000, XP and 2003. You need to configure IIS to run ASP.NET Web applications. You should configure IIS prior to the installation of Visual Studio .NET software on your machine to avoid errors. In most cases configuring IIS after the installation of VS .NET will result in many errors and unexpected behaviour by the application.

By default, IIS creates a folder on the server's hard drive with the name Inetpub. The Inetpub folder contains a subfolder called wwwroot. The wwwroot folder is the root for the Web site. All the ASP.NET applications you develop using VS .NET are saved in this wwwroot folder.

Web Hosting

You also can test and run your applications on a server owned by hosting providers. The host will give you details you need to know to upload files onto his server, test those files, etc. Web Hosting providers charge some amount for providing service. There are some hosting providers who provide some space for a certain period of time on their servers for ASP.NET developers to test their applications free of charge. You can find about them on the resources page of this site.

Visual Studio .NET

Visual Studio .NET consists of five CDs. Follow the guide lines on installing the software as mentioned in the manuals/ files accompanying the software.

Database

To develop ASP.NET database applications you need to install SQL Server 2000 or higher or Oracle depending on the database you wish to use. Use of SQL Server with ASP.NET is recommended as it's said that SQL Connections are 70% faster than OLEDB Connections. Also, performance improves dramatically when you use SQL Server with ASP.NET.

Self Learning Exercise-3

i. Fill in the Blanks

b) ASP.NET is based on the _______, class libraries and other tools which are integrated into the .NET Framework

12.5 Web Application State

A Web page is recreated every time it is posted back to the server. In traditional Web programming this means that all the information within the page and information in the controls is lost with each round trip. To overcome this limitation of traditional Web programming, the ASP.NET page framework includes various options to help us preserve changes. One option involves keeping information on the client, directly in the page or in a cookie and another option involves storing information on the server between round trips. We will take a look at both the options.

Saving State in the Client

If we decide to save data in the client then that data is not stored on the server. This means that the page has to store all the data in controls so that it can be sent back to the server when the page is posted back to the server. The different ways in which you can save state in a client are discussed below:

HTML Hidden Form Fields

The default way of saving the state of the data is to use HTML hidden fields. A hidden field stores text data with the HTML <input style="hidden">. A hidden field will not be visible in the browser but we can set its properties just like we set properties for other standard controls. When a page is posted to the server, the content of a hidden field is sent in the HTTP Form collection along with the values of other controls. In order for hidden field values to be available during page processing, we must submit the page using an HTTP post method. That is, you cannot take advantage of hidden fields if a page is processed in response to a link or HTTP GET method.

Cookies

A cookie is a small text file that stores data on the client's machine or is persistent in-memory during the client browser session. It contains page-specific information the server sends to the client along with page output. Cookies can be temporary with specific expiration times and dates or persistent. We can use cookies to store information about a particular client, session, or application. Cookies are saved on client machine, and when the browser requests a page, it sends the information in the cookie along with the request information. The server can read the cookie and extract its value. Cookies are a relatively secure way of maintaining user-specific data as the browser can only send the data back to the server that originally created the cookie.

Query Strings

A query string is information added to the end of a page's URL. For example, it looks like the following:

https://login.yahoo.com/config/login_verify2?src=ym

In the above URL , the query string starts with the question mark (?) and includes attribute-value pair, called "src." Query strings provide a simple and limited way of maintaining state information. Most browsers impose a 255-character limit on the length of the URL. Also, as the query values are exposed to the Internet via the URL, in some cases security may be an issue. In order for query string values to be available during page processing, we must submit the page using an HTTP get method. You cannot take advantage of a query string if a page is processed in response to an HTTP post method.

View State

The Control.ViewState property provides a way for retaining values between multiple requests for the same page. This is the method that the page uses to preserve page and control property values between round trips. When a page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field. When the page is posted back to the server, the page parses the view state string at page initialization and restores property information in the page.

Saving State on the Server

We can also save an application's state on the server. Saving an application's state on the server provides with more security than client-side options. The different ways in which you can save state in a server are discussed below:

Application State

An application's state is stored using the application's state object (HttpApplicationState class) for each active Web Application. Application state is a global storage mechanism accessible from all pages in the Web application and is useful for storing information that needs to be maintained between server round trips and between pages. Application state is a key-value dictionary structure created during each request to a specific URL. You can add your application-specific information to this structure to store it between page requests. Once you add your application-specific information to application state, the server manages it for you.

Session State

Besides application state, ASP.NET also can store session states using a session state object (HttpSessionState class) for each active Web Application. Session state is similar to application state, but it is scoped to the current browser session. If different users are using an application, each user will have a different session state. If a user leaves the application and returns later, that user will have a different session state. Session state is a key-value dictionary structure for storing session-specific information that needs to be maintained between server round trips and between requests for pages. Once we add our application-specific information to session state, the server manages this object for us. Depending on the options we specify, session information can be stored in cookies, an out-of-process server, or a SQL Server.

Using Database

Maintaining state using database is a very common method when storing user-specific information where the information store is large. Database storage is particularly useful for maintaining long-term state or state that must be preserved even if the server must be restarted. The database approach is often used along with cookies. For example, when a user accesses an application, he might need to enter a username/password to log in. You can look up the user in your database and then pass a cookie to the user. The cookie might contain only the ID of the user in your database. You can then use the cookie in the requests that follow to find the user information in the database as needed.

Self Learning Exercise-4

i. Fill in the Blanks

c) ________ is a small text file that stores data on the client's machine or is persistent in-memory during the client browser session.

d) The __________________property provides a way for retaining values between multiple requests for the same page.

ii. State True or False

d) A Web page is recreated every time it is posted back to the server

12.6 Web Forms

Web Forms are based on ASP.NET. Working with Web Forms is similar to working with Windows Forms. But the difference is that we will create Web pages with Web forms that will be accessible by a Web browser. Web Forms are Web pages that serve as the user interface for a Web application. A Web Forms page presents information to the user in any browser or client device and implements application logic using server-side code. Web Forms are based on the System.Web.UI.Page class. The class hierarchy for the page class is shown below.

```
Object
        Control
                TemplateControl
                                Page
```

12.6.1 Components of Web Forms

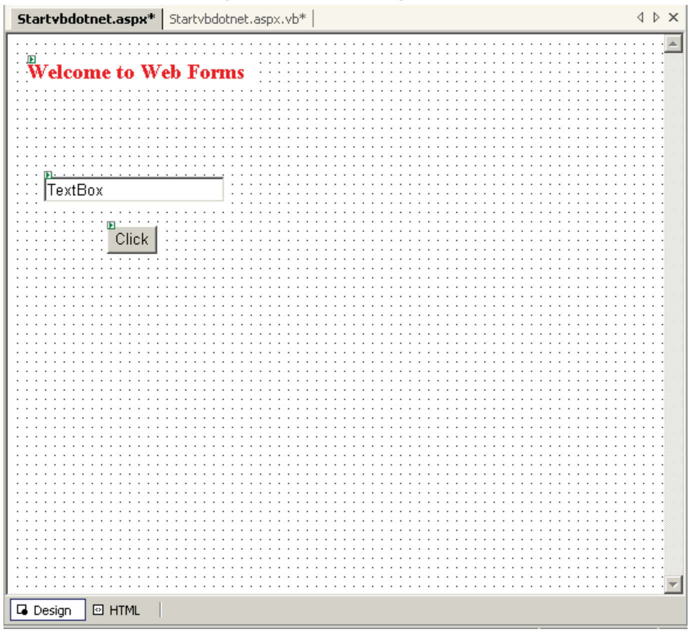
In Web Forms pages, the user interface programming is divided into two parts: the visual component (design page) and the logic (code behind page).

The visual element is the Web Forms page. The page consists of a file with static HTML, or ASP.NET server controls, or both simultaneously. The Web Forms page works as a container for the static text and the controls we want to display. Using the Visual Studio Web Forms Designer and ASP.NET server controls, we can design the form just like in any Visual Studio application.

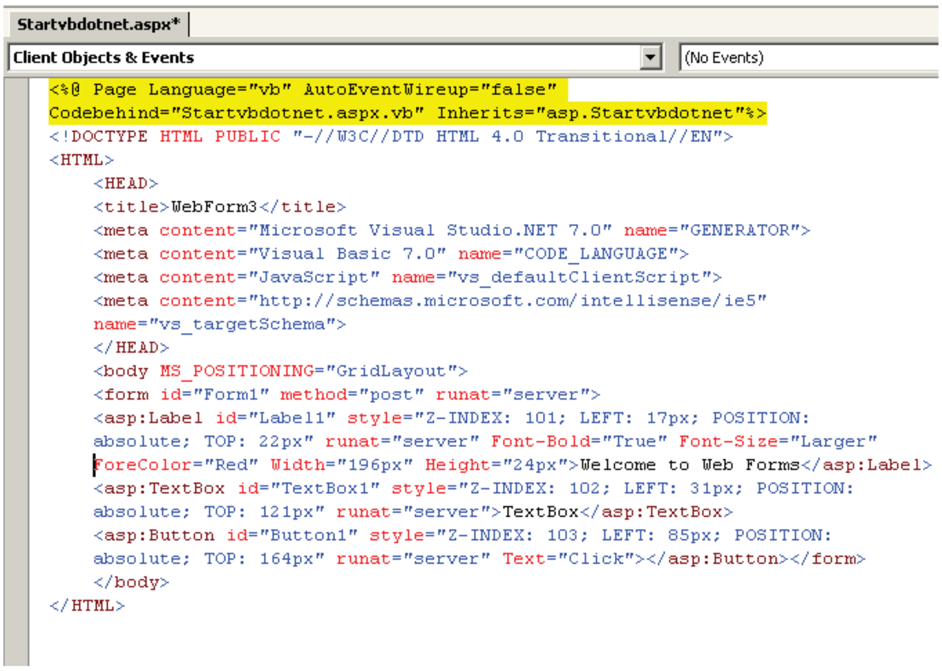
The logic for the Web Forms page consists of code that we create to interact with the form. The programming logic is in a separate file from the user interface file. This file is the "code-behind" file and has an ".aspx.vb" (VB) or ".aspx.cs" (C-Sharp) extension. The logic we write in the code-behind file can be written in Visual Basic or Visual C#.

The code-behind class files for all Web Forms pages in a project are compiled into the project dynamic-link library (.dll) file. The .aspx page file is also compiled, but differently. The first time a user loads the aspx page, ASP.NET automatically generates a .NET class file that represents the page, and compiles it to a second .dll file. The generated class for the aspx page inherits from the code-behind class that was compiled into the project .dll file. When the user requests the Web page URL, the .dll files run on the server and dynamically produces the HTML output for your page.

When you open a new ASP.NET Web Application in Visual Studio .NET the form that is loaded (WebForm1.aspx) looks like the image shown in Figure 12.1.



The form opens in design mode and you can switch to HTML view by clicking on the HTML tab. Startvbdotnet.aspx file in the image above is standard HTML with ASP elements embedded in it.



```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="Startvbdotnet.aspx.vb" Inherits="asp.Startvbdotnet"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
    <title>WebForm3</title>
    <meta content="Microsoft Visual Studio.NET 7.0" name="GENERATOR">
    <meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
    name="vs_targetSchema">
    </HEAD>
    <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
    <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 17px; POSITION:
    absolute; TOP: 22px" runat="server" Font-Bold="True" Font-Size="Larger"
    ForeColor="Red" Width="196px" Height="24px">Welcome to Web Forms</asp:Label>
    <asp:TextBox id="TextBox1" style="Z-INDEX: 102; LEFT: 31px; POSITION:
    absolute; TOP: 121px" runat="server">TextBox</asp:TextBox>
    <asp:Button id="Button1" style="Z-INDEX: 103; LEFT: 85px; POSITION:
    absolute; TOP: 164px" runat="server" Text="Click"></asp:Button></form>
    </body>
</HTML>
```

From the above image notice the Codebehind attribute (second line). The codebehind attribute connects this code to the appropriate Visual Basic or C# code (code behind file). Startvbdotnet.aspx.vb, the codebehind file for Startvbdotnet.aspx where you write your logic looks like this:

```
Public Class Startvbdotnet Inherits System.Web.UI.Page
Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
Protected WithEvents Button1 As System.Web.UI.WebControls.Button
Protected WithEvents Label1 As System.Web.UI.WebControls.Label
#Region " Web Form Designer Generated Code "
'This call is required by the Web Form Designer.
Private Sub InitializeComponent()
End Sub
Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles MyBase.Init
'CODEGEN: This method call is required by the Web Form Designer
'Do not modify it using the code editor.
InitializeComponent()
End Sub
#End Region
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles MyBase.Load
'Put user code to initialize the page here
End Sub
End Class
```

12.6.2 Notable Properties of Page Object

Below are some notable properties of Page objects:

Application	:	Gets an application object
ClientTarget	:	Gets/Sets if you want to override automatic browser capabilities detection and handle page rendering for specific browsers
ErrorPage	:	Gets/Sets an error page's URL in case there are unhandled page exceptions
IsPostBack	:	Indicates if a page was created after a client postback or if it is being loaded for the first time
IsValid	:	Indicates if the page validation was successful Request Gets the current http request object
Response	:	Gets the current http response object
Server	:	Gets the current server object
Session	:	Gets the current session object

Site	:	Gets Web Site data
User	:	Gets data about the user
Validators	:	Gets a collection of validation controls on the page

12.6.3 The ASPX Extension

Many of us wonder why the extension for ASP.NET is .aspx. Well, long time ago, when ASP.NET was being developed at Microsoft it was referred to as ASP+ (ASP Plus). You can't use a "+" symbol in a filename but if you turn the + symbol about 45 degrees, it looks like a x. Microsoft chose .aspx as the extension of ASP+. After the name was changed to ASP.NET, Microsoft didn't change the extension and left it as aspx.

Self Learning Exercise-5

i. Fill in the Blanks

e) Web Forms are based on the ________ class.

f) In Web Forms pages, the user interface programming is divided into two parts: the visual component and the _________.

12.7 Web Forms User Controls

In addition to HTML and Web server controls, you can simply create your own custom, reusable controls by using the same techniques you have learned to develop Web Forms pages. These controls are called user controls.

User controls offer you an easy way to partition and reuse common user interface (UI) functionality across your ASP.NET Web applications. Like a Web Forms page, you can author these controls with any text editor, or develop them using code-behind classes. Also, like a Web Forms page, user controls are compiled when first requested and stored in server memory to reduce the response time for subsequent requests. Unlike pages, however, user controls cannot be requested independently; they must be included in a Web Forms page to work.

User controls offer you greater flexibility than server-side includes (SSIs) by accessing the object model support provided by ASP.NET. Rather than simply including the functionality provided by another file, you can program against any properties you declare in the control, just like any other ASP.NET server control.

While you need to choose a single language when authoring a user control, you can include multiple user controls in a single Web Forms page that have been authored in different languages. For example, you can create a user control with Visual Basic that imports data from an XML file and another user control, created with C#, that contains an order form, and include both controls in the same Web Forms page.

When you create a Web application using Visual Studio .NET, all pages and user controls in the application must be in the same programming language.

In addition, you can cache output from a user control independently from the rest of its containing Web Forms page. Called fragment caching, this technique can improve performance for your site if used appropriately. For example, if your user control contains an ASP.NET server control that makes a database request, but the rest of the page contains only literal text and simple code that runs on the server, you can fragment cache the user control to increase your application's performance.

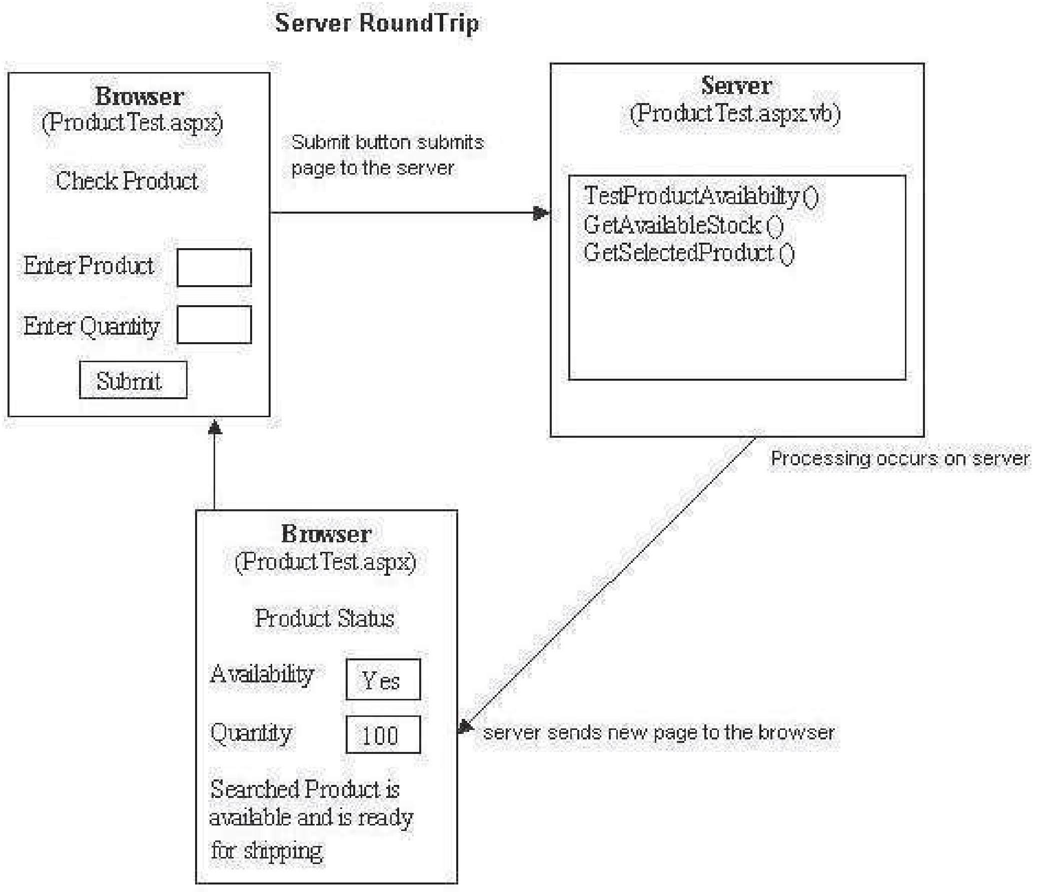
12.8 Web Forms Page Life Cycle

In general, the life cycle for a Web Forms page is similar to that of any Web process that runs on the server. Certain characteristics of Web processing information is passed via HTTP protocol, the stateless nature of Web pages, and so on. However, the ASP.NET page framework performs many Web application services for us. It is important to understand the sequence of events that occur when a Web Forms page is processed.

It will be helpful to understand some fundamental characteristics of how Web Forms pages work in Web applications before we examine the details of what goes on inside a page when it is processed.

12.8.1 Round Trips

Most Web pages require processing on the server. For example, consider a products page used to check the availability of a certain product. When a user selects his product and hits the submit button the page must check on the server to see whether the selected product is available or not. This kind of functionality is achieved by handling server control events. Whenever a user interaction requires processing on the server, the Web page is posted back to the server, processed, and is returned back to the browser. This sequence is called round trip. The image below demonstrates server round trip.



In any Web scenario, Web pages are recreated with every round trip. When the server finishes processing and sends the page to the browser, it discards the page information. This frees server resources after each request and a Web application can scale to support hundreds or thousands of simultaneous users. The next time the page is posted, the server starts over in creating and processing it, and for this reason, Web pages are said to be stateless. Stateless means the values of a page's variables and controls are not saved on the server.

ASP.NET works around the above said limitations in the following ways:

- ASP.NET saves page and control properties between round trips. This is referred to as saving the view state of the control.
- It provides state management facilities so that you can save your own variable and application-specific or session-specific information between round trips.
- It can detect when a form is requested for the first time versus when the form is posted, and allows you to program accordingly. You may want a different behavior during a page postback versus an initial request.

12.8.2 Stages in Web Forms Processing

Table 12.1: Web Forms Processing Stages

Stage	**Means**	**Use**
Page Initialization	The page's Page_Init event is raised, and the page and control view state are restored.	During this event, the ASP.NET page framework restores the control properties and postback data.
User Code Initialization	The page's Page_Load event is raised.	Read and restore values stored previously, Using the Page.IsPostBack property, check whether this is the first time the page is being processed. If this is the first time the page is being processed then perform initial data binding. Otherwise, restore control values. Read and update control properties.
Validation	The Validate method of any validator Web server controls is invoked to perform the control's specified validation.	Test the outcome of validation in an event handler
Event Handling	If the page was called in response to a form event, the corresponding event handler in the page is called during this stage	Perform application-specific processing and handle the specific event raised.
Cleanup	The Page_Unload event is called because the page has finished rendering and is ready to be discarded.	Perform final cleanup work. Close files, closing database connections and discard objects.

[source: http://www.startvbdotnet.com]

Self Learning Exercise-6

i. Fill in the Blanks

g) Cleanup stage of page cycle called the __________event because the page has finished rendering and is ready to be discarded.

ii. State True or False

e) ASP.NET saves page and control properties between round trips. This is referred to as saving the view state of the control.

12.9 Conditional Statements and Loops

12.9.1 If....Then....Else Statement

If conditional expression is one of the most useful control structures which allows us to execute a expression if a condition is true and execute a different expression if it is False. If the condition is true the statements following the Then keyword will be executed, else the statements following the ElseIf will be checked and if true, will be executed, else the statements in the else part will be executed.

If Then Sample

The following code demonstrates If..Then..Else conditional statement. It asks the user to enter a number between 1 and 3 in a textbox and checks for the number entered using the If..Then..Else statement and displays some text if the condition is true.

```
Private Sub IfThen_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles IfThen.Click
If CInt(TextBox1.Text) = 1 Then
Label1.Text = "You entered" & CInt(TextBox1.Text)
ElseIf CInt(TextBox1.Text) = 2 Then
Label1.Text = "You entered" & CInt(TextBox2.Text)
ElseIf CInt(TextBox1.Text) = 3 Then
Label1.Text = "You entered" & CInt(TextBox2.Text)
Else
Label1.Text = "You entered another number"
End If
End Sub
```

12.9.2 Select Case Statement

Select Case is similar to If..Then..Else statement. Select Case allows us to do various comparisons against one variable that we use throughout the whole statement. Select Case is cleaner and easier to understand if you are continually comparing against one variable.

Select Case Sample

The following sample asks the user to enter a vowel in a textbox and checks the value entered using the the Select Case statement and if true displays some text.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
```

```
Select Case TextBox1.Text
Case “a”
Label1.text = “You entered A”
Case “e”
Label1.Text = “You entered E”
Case “i”
Label1.Text = “You entered I”
Case “o”
Label1.Text = “You entered O”
Case “u”
Label1.Text = “You entered U”
End Select
End Sub
```

12.9.3 For Loop

The For loop is the most popular loop. For loops enables us to execute a series of expressions multiple numbers of times. The For loop needs a loop index, which counts the number of loop iterations as the loop executes.

For Loop Sample

The following code demonstrates for loop.

```
Private Sub ForLoop_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles ForLoop.Click
‘assuming you have a button named ForLoop
Dim i As Integer
For i = 1 To 5
Response.Write(“Welcome to ASP.NET”)
Next i
End Sub
```

12.9.4 Do Loop

The Do loop can be used to execute a fixed block of statements indefinite number of times. The Do loop keeps executing it’s statements while or until the condition is true. Two keywords while and until can be used with the do loop. The Do loop also supports an Exit Do statement which makes the loop to exit at any moment.

Do Loop Sample

The following code demonstrates Do loop.

```
Private Sub DoWhile_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles DoWhile.Click
‘assuming you have a button named DoWhile
Dim x As Integer
```

```
x = 0
Do Until x = 10
x = x + 1
Response.Write(x)
Loop
End Sub
```

12.10 Arrays and Strings

Arrays

An array is a way of declaring a whole group of variables at once. Arrays are programming constructs that store data and allow us to access them by numeric index or subscript. Arrays help us create shorter and simpler code in many situations. Arrays are based on the System.Array namespace. They are declared using Dim, ReDim, Static, Private, Public and Protected keywords. An array can have one dimension or more than one. The dimensionality of an array refers to the number of subscripts used to identify an individual element.

An array declaration looks as follows: **Dim Sports (5) as String**. This single line creates a Sports array that has 6 elements starting from Sports(0) to Sports(5). The first element of an array is always referred by zero index. To change the dimension (redimension) of the Sports array which we already declared we use the following statement: ReDim Sports(20). The size of Sports array changes from 11 to 21.

Arrays Sample

The following sample will demonstrate arrays. Drag a button and a textbox control on to the Web Forms page. Open the code designer window and paste the following code.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
Dim Sports(5) As String
Sports(0)=”Tennis”
Sports(1) = “Cricket”
Sports(2) = “Rubgy”
Sports(3) = “Aussie Rules”
Sports(4) = “Soccer”
Sports(5) = “Hockey”
Label1.Text = “The sport in the location you entered is” & “ “ &_
Sports(CInt(TextBox1.Text))
End Sub
```

The above sample code creates a Sports array with six elements, asks the user to enter a number between 0 and 5 and displays the sport that is stored in the location (number) the user entered on a label when the button is clicked. Test the code at the bottom of this page for understanding.

Strings

Strings are supported by the .NET String class. The String data type can represent a series of characters and can contain approximately up to 2 billion Unicode characters. There are many built-in functions in the String class. Some .NET Framework functions are also built into the String class. Some common String functions are discussed below:

LCase	:	Converts a string to lower case
UCase	:	Converts a string to upper case
Len	:	Calculates the number of characters in the string
LTrim	:	Omits spaces that appear on the left-side of the string
RTrim	:	Omits spaces that appear on the right-side of the string
Trim	:	Omits both leading and trailing spaces
Clone	:	Clones a string
Concat	:	Joins two strings
Space	:	Used to create a string with spaces
Left	:	Takes two arguments and returns a string that consists of the left-most characters of the string sent
Right	:	Takes two arguments and returns a string that consists of the right-most characters of the string sent
Instr	:	Searches and returns a shorter string within a long string
Replace	:	Seraches for a small string in a long string and replaces it with the specified string

Self Learning Exercise-7

i. Fill in the Blanks

h) Strings are supported by the .NET _______class.

ii. State True or False

f) Select Case allows us to do various comparisons against many variables that we use throughout the whole statement.

g) Arrays are programming constructs that store data and allow us to access them by numeric index or subscript.

12.11 Functions

Variables

Variables are used to store data. A variable has a name to which we refer and the data type, the type of data the variable holds. You need to declare variables before using them. Variables are declared with the Dim keyword and Dim stands for Dimension.

Variables Sample

The following code creates two variables, adds them and stores the sum in a third variable.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
Dim x As Integer = 10
Dim y As Integer = 20
Dim z As Integer = x + y
Response.Write("Sum is" & " " & z)
End Sub
```

Constants

When we have certain values that we frequently use while programming, we should use Constants. A value declared as constant is of fixed value that cannot be changed once set. Constants should be declared as Public if we want it to be accessed by all parts of the application. We use the Const keyword to declare a constant. The following line of code declares a constant:

```
Public Const Pi as Double=3.14159265
```

SubProcedures

Procedures are a series of statements that are executed when called. Procedures makes us handle the code in a simple and organized fashion. Sub procedures are procedures which do not return a value. Each time when the Sub procedure is called, the statements within it are executed until the matching End Sub is encountered. The Page_Load event, which is the starting point of the program itself is a sub procedure. When the application starts execution, the control is transferred to Page_Load procedure automatically which is called by default.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
Display()
'control looks for a sub procedure named Display() and executes it
End Sub
Sub Display()
Response.Write("In Sub Procedure")
End Sub
```

Functions

Function is a Procedure which returns a value. Functions are used to evaluate data, make calculations, or to transform data. Declaring a Function is similar to declaring a Sub procedure but functions are declared with the Function keyword. The following is an example on Functions. This simple application is an online loan repayment calculator that asks the user to enter the amount he wants to borrow, the duration of the loan and based on that information calculates his monthly repayments. To start, drag four labels, two textboxes and a button from the toolbox on to the Web Forms page. Open the code designer and paste the following code. Look at Live Code demo towards the botom of this page for user interface design.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
Calc()
'when the user submits his data, control passes to Calc function
End Sub
Public Function Calc() As Long
Dim amt, temp, temp1, totinterest, mi, mitemp, tot As Double
```

```
'declaring the required variables
temp = CInt(TextBox1.Text) * 13
'multiplying the amount the user wants to borrow with 13.Assuming the
'interest rate to be 13%
totinterest = temp / 100
'calculating the interest amount for the amount entered
mitemp = CInt(TextBox2.Text) * 12
'multipling the years entered with 12 to get the number of months
mi = totinterest / mitemp
'calculating monthly interest
temp1 = CInt(TextBox2.Text) * 12
'multipling the years entered with 12 to get the number of months
amt = CInt(TextBox1.Text) / temp1
'dividing the amount borrowed with number of months it is borrowed for
tot = amt + mi
'calculating the total by adding the amount + monthly interest
Label1.Text = "Your monthly repayment is:" & " " & Int(tot)
disp()
End Function
Public Sub Disp()
        Label4.Text = "Not Satisfied? Would you like to try another option?"
End Sub
```

Self Learning Exercise-8

ii. State True or False

h) Variables are used to store data.

i) Function is a Procedure which returns a value.

12.12 Creating and Running a Web Application

The following is a simple and interesting sample. It asks the user to enter some information and generates a username and password based on the information entered and displays the newly generated login information to the user. To start, drag six labels, four textboxes and a button control on to the Web Forms page. TextBox1 accepts first name, TextBox2 accepts last name, TextBox3 accepts age and TextBox4 accepts country. Look at the sample below for the user interface. Open the code designer window and paste the following code.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
Dim mystr As String
Dim mystr1 As String
Dim mystring As String
Dim midd As String
'declaring three strings
mystr = Left(TextBox1.Text, 3)
'using the left function
mystr1 = Right(TextBox2.Text, 3)
'using the right function
midd = Mid(TextBox4.Text, 2, 6) & CInt(TextBox3.Text) - 11
'using the mid function
mystring = mystr & mystr1 & TextBox3.Text
Label5.Text = "Your user name is" & " " & mystring
Label6.Text = "Your random password generated by our server is" & " " & midd
TextBox1.Visible = False
TextBox2.Visible = False
TextBox3.Visible = False
TextBox4.Visible = False
Label1.Visible = False
Label2.Visible = False
Label3.Visible = False
Label4.Visible = False
End Sub
```

To run this application press F5. you will se the output .Other wise you can also run this application on internet explorer on typing http://localhost or the name of server and the path where you have stored the data.

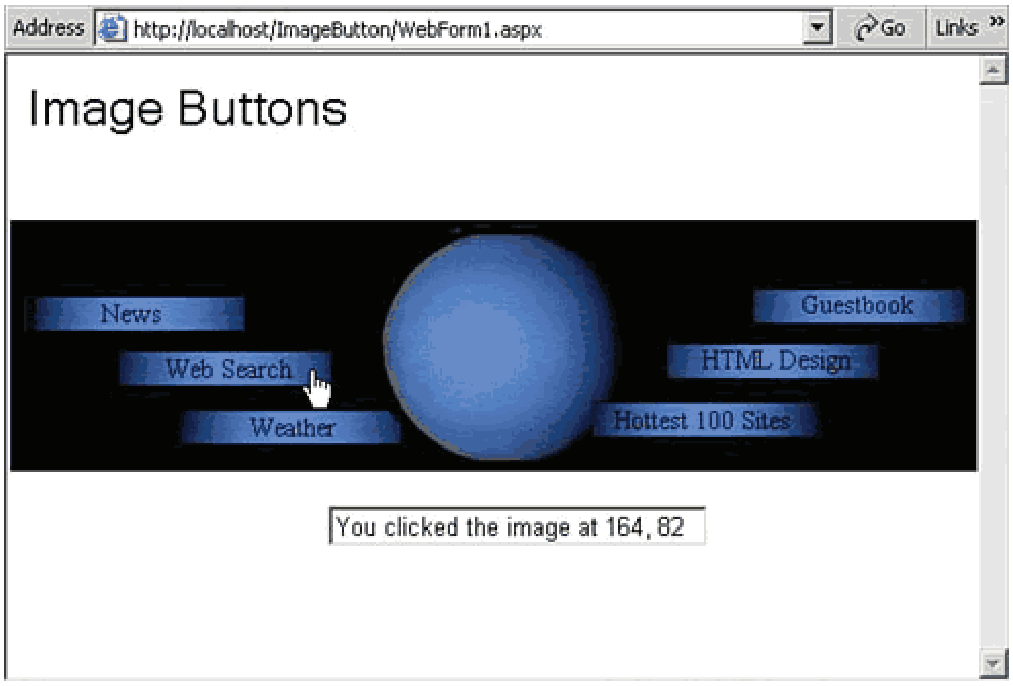
You can also note the path when you run this application in ASP environment.

12.13 Multiform Web Project

12.13.1 Using Image Buttons

Image controls let you display static images, but not all images are static in Web pages; some are interactive. For example, Web pages can support image maps—those clickable images with "hotspots" that, when clicked, will cause something to happen, as well as images that act like buttons when you click them. You can support those kinds of interactive images with image buttons.

Image controls are supported in HTML with HTML <img> elements, but image buttons use HTML <input> elements where the type attribute is set to "image" (in other words, an HTML image map). As with other Web server controls, image buttons support both Click and Command events. You can see an image button at work in Figure 14.2, where the ImageButton example in the code for this book is displaying the location of the mouse when you click the image.



You clicked the image at 164, 82

Here's the HTML created to support the image button you see in Figure. The image here is stored in the file map.jpg.

```
<input type="image" name="ImageButton1" id="ImageButton1"
src="file:///C:\inetpub\wwwroot\ImageButton\map.jpg" border="0"
style="Z-INDEX: 102; LEFT: 2px; POSITION: absolute; TOP: 90px" />
```

Image buttons are supported with the System.Web.UI.WebControls.ImageButton class; here is the hierarchy of this class:

```
System.Object
 System.Web.UI.Control
  System.Web.UI.WebControls.WebControl
   System.Web.UI.WebControls.Image
    System.Web.UI.WebControls.ImageButton
```

You can find the significant public properties of System.Web.UI.WebControls.ImageButton objects in Table 12.2 and the significant public events in Table 12.3. (This class has no non-inherited methods.) Note that as with other Web server controls, these tables do not list the significant properties, methods, and events this class inherits from the Control and WebControl classes this class inherits from the System.Web.UI.WebControls.Image class

Table 12.2 Significant Public Properties of ImageButton Objects

Property	Means
CommandArgument	Returns or sets an (optional) value holding text associated with the command given by the CommandName property.
CommandName	Returns or sets the command name for this image button. If you assign a value to this property, the Command event occurs when the button is clicked.

Table 12.3 shows the significant Public Events of ImageButton Objects

Method	Means
Click	Occurs when the image button was clicked.
Command	Occurs when the image button was clicked and the CommandName property holds some text—use a Command event handler to handle this one.

Image buttons are images that also support Click events. When you handle Click events in image buttons, you are passed the location of the mouse in the image in an ImageClickEventArgs object; that location is what you need when you want to create an image map that the user can click to perform some action.

The position of the mouse is stored in pixels; the origin, (0, 0), is at the upper-left corner of the image. Here's how the ImageButton example displays the coordinates in the image at which the user clicked the mouse:

```
Private Sub ImageButton1_Click(ByVal sender As System.Object, _
ByVal e As System.Web.UI.ImageClickEventArgs) Handles _
ImageButton1.Click
 TextBox1.Text = "You clicked the image at " & e.X & ", " & e.Y
End Sub
```

You can see the results in Figure. Image maps often let the user navigate to a new URL when you click a "hotspot" in them, and you can handle that with the Response object's Redirect method. Here's an example that makes the browser navigate to http://www.microsoft.com when the user clicks a region of the image—the rectangle stretching from (100, 50) to (200, 150)—which we'll treat as a hotspot:

```
Private Sub ImageButton1_Click(ByVal sender As System.Object, _
ByVal e As System.Web.UI.ImageClickEventArgs) Handles _
ImageButton1.Click
 If (e.X >= 100 And e.X <= 200) And (e.Y >= 50 And e.Y <= 150) Then
  Response.Redirect("http://www.microsoft.com")
 End If
End Sub
```

You can also use the Command event handler to make the ImageButton control work like command buttons. In particular, you can connect a command name to the image button with the CommandName property, and the CommandArgument property can also be used to pass additional information about the command. Here's how you might put that to work in code:

```
Private Sub ImageButton1_Command(ByVal sender As Object, _
ByVal e As System.Web.UI.WebControls.CommandEventArgs) _
Handles ImageButton1.Command
 If e.CommandName = "NavigateButton" Then
  TextBox1.Text = e.CommandArgument
 End If
```

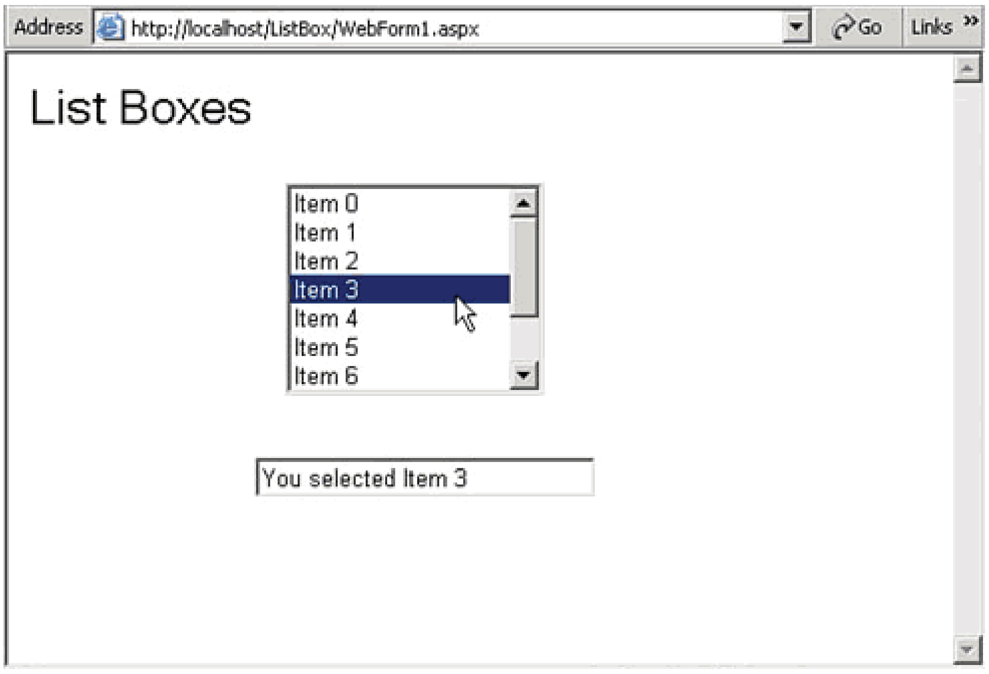
End Sub

As with image controls, you can set the URL of the image to be used to the ImageUrl property. To set the ImageUrl property at design time, click this property in the Properties window and browse to the image you want to use. You can also set the ImageUrl property at runtime; just assign it a string containing the URL of the image you want to use. You can also set the image's width and height with the Width and Height properties.

That's it for image controls and buttons. There are only a few ways to display images in Web pages that browsers will understand, and we've covered them now—the background property of Web forms that let you set background images, image controls that translate into HTML <img> elements, and image buttons that translate into HTML <input type = "image"> elements. Next, we'll take a look at Web server list controls.

12.13.2Using List Boxes

As when working with Windows forms, you use list boxes to create a control that allows single or multiple selection of items from a list. When the user clicks an item in the list box, that item appears in the text box at the bottom of the application, as you see in the figure.



In HTML, list boxes are supported with the HTML <select> control, and the items in the list boxare supported with HTML <option> elements. For example, here's the HTML that creates the single-selection list box.

<select name="ListBox1" size="4" onchange="__doPostBack('ListBox1',")"

language="javascript" id="ListBox1"

style="height:128px;width:145px;Z-INDEX: 102;

LEFT: 156px; POSITION: absolute; TOP: 72px">

Self Learning Exercise-9

i. Fill in the Blanks

i) ______________ Returns or sets an (optional) value holding text associated with the command given by the CommandName property

12.14 Page Class

Namespace: System.Web.UI

Assembly: System.Web (in system.web.dll)

Syntax

Public Class Page Inherits TemplateControl Implements IHttpHandler

The Page class is associated with files that have an .aspx extension. These files are compiled at run time as Page objects and cached in server memory.

If you want to create a Web Forms page using the code-behind technique, derive from this class. Rapid application development (RAD) designers, such as Microsoft Visual Studio, automatically use this model to create Web Forms pages.

The Page object serves as the naming container for all server controls in a page, except those that implement the INamingContainer interface or are child controls of controls that implement this interface.

The Page class is a control that acts as the user interface for your Web application, and as such should be scrutinized to make sure best practices for writing secure code and securing applications are followed.

The following code example demonstrates how the Page class is used in the code-behind page model. Note that the code-behind source file declares a partial class that inherits from a base page class. The base page class can be Page, or it can be another class that derives from Page. Furthermore, note that the partial class allows the code-behind file to use controls defined on the page without the need to define them as field members.

```
<%@ Page Language=”VB” CodeFile=”pageexample.aspx.vb” Inherits=”MyCodeBehindVB”
%>
<html>
 <head runat=”server”>
 <title>Page Class Example</title>
 </head>
 <body>
  <form runat=”server”>
 <div>
 <table>
 <tr>
 <td> Name: </td>
 <td> <asp:textbox id=”MyTextBox” runat=”server”/> </td>
 </tr>
 <tr>
 <td></td>
```

```
<td><asp:button id=”MyButton” text=”Click Here” onclick=”SubmitBtn_Click” runat=”server”/
></td>
</tr>
<tr>
<td></td>
<td><span id=”MySpan” runat=”server” /></td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

Self Learning Exercise-10

i. Fill in the Blanks

j) The ________ object serves as the naming container for all server controls in a page.

12.15 Summary

In this unit you have learned the features of ASP.NET, its server side and client side Technologies, the life cycle of a web page, steps in creation and running a web application and a brief introduction of System.Web.UI.Page class.

12.16 Glossary

CLR	Common Language Runtime provided by the .NET Framework. This CLR manages execution of the code we write.
IIS	Internet Information Server. To develop a Web Application IIS is required on the machine.
XML Web Services	XML Web services allow applications to communicate and share data over the Internet, regardless of operating system or programming language

12.17 Further Readings and References

1. Mridula Parihar, Essam Ahmed, Jim Chandler, Bill Hatfield ,”ASP.NET Bible”, Unique Color carton, New Delhi
2. Mathew Macdonald, “ASP.NET: The Complete Reference”, Tata Mcgraw Hill, New Delhi
3. John Alexander, Billy Hollis, “Developing Web Applications with Visual Basic .NET and ASP.NET”, John Wiley New Delhi
4. http://www.startvbdotnet.com
5. http://www.msdn2.microsoft.com

12.18 Answers to Self Learning Exercises

i. Fill in the Blanks	ii. True/False
a) server-side	a) True
b) CLR	b) False
c) Cookie	c) True
d) Control.ViewState	d) True
e) System.Web.UI.Page	e) True
f) Logic	f) False
g) Page_Unload	g) True
h) String	h) True
i) CommandArgument	i) True
j) Page	

12.19 UNIT END QUESTIONS

1. What is the differences between ASP.NET and Client-Side Technologies
2. List out the features of ASP.NET
3. What do you understand by ASP.NET Development Environment ? Explain.
4. Explain the Life Cycle of web form page.
5. What is the use of System.Web.UI.Page Class

Unit - 13: Controls in VB.NET

Structure of the Unit

13.0 Objective

After going through this unit you will be able to,

- Understand different types of controls – HTML & Server controls
- Use of dialog boxes for interaction with user
- Understand services of web pages & different operations performed by them
- Apply the process of deployment and installation of an application.

13.1 Introduction

From the previous Unit you have learned about ASP.NET, the web form and their components and the life cycle of a web page. You have also learned the concept of functions, arrays etc.

In this Unit we will discuss about how to make a web pages in .NET ,controls that can be placed on a web page and different services performed by web page.

Before going through this unit it is required that you must read Unit-12 and solve the exercises.

13.2 Web Server Controls

Web server controls are one of the coolest new things about ASP.NET. I tend to think of them as HTML controls on steroids, but let me introduce you to some of them and see what you think.

Web server controls are just like HTML controls... only different. Let me clarify that. Web controls are easily identified by the fact that they have a runat=”server” attribute. They are also handled at the server by the ASP.NET runtime when a page containing them is requested. They can expose and raise server events which you can use to interact with them. They are identified by giving them an id attribute which you can then use to reference the control in your code. In all these ways, web controls are just like HTML controls.

The most obvious difference between the two is that the tags for web controls don’t look like HTML tags. This is because they are not as closely tied to the basic HTML tags as HTML controls are. While HTML controls tend to emit their corresponding HTML tag (with the attributes you’ve set), they almost always just emit that HTML tag. Web controls can emit multiple HTML tags in all sorts of combinations (or whatever else is needed) in order to accomplish their task. They perform higher level fuctions and do not necessarily map directly to any one HTML tag. Their object model is generally more complex and is ususally more abstract then that of an HTML control.

They look something like this:

```
<asp:label id=”lblSample” runat=”server” />
```

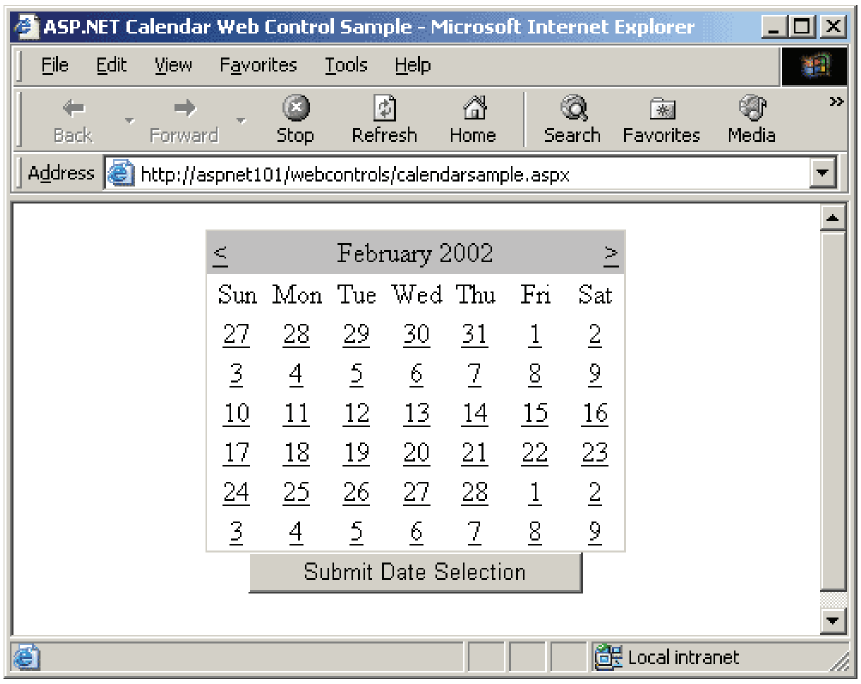
That was simply a label control. It doesn’t do much except output whatever text you assign to it surrounded by a <span> tag. It’s one of the simplest of all the web controls. I’ve included a sample aspx file named labelsample.aspx illustrating the above line in use in this lesson’s zip file available at the bottom of this page.

calendarsample.aspx

```
<%@ Page Language=”VB” %>
<script runat=”server”>
  Sub btnSubmit_Click(Sender As Object, E As EventArgs)
    If calSample.SelectedDates.Count = 0 Then
      frmDateSelection.Visible = True
```

```
        lblSelectedDate.Visible = False
      Else
        frmDateSelection.Visible = False
        lblSelectedDate.Visible = True
        lblSelectedDate.Text = "You Selected: " _
            & calSample.SelectedDate
      End If
   End Sub
</script>
<html>
<head>
<title>ASP.NET Calendar Web Control Sample</title>
</head>
<body bgcolor="#FFFFFF">
<div align="center">
<form id="frmDateSelection" action="calendarsample.aspx"
  method="post" runat="server">
<asp:calendar id="calSample" runat="server" />
<asp:button id="btnSubmit" text="Submit Date Selection"
  onClick="btnSubmit_Click" runat="server" />
</form>
<asp:label id="lblSelectedDate" runat="server" />
</div>
</body>
</html>
```

It declares a calendar web control.


ASP.NET Calendar Web Control Sample - Microsoft Internet Explorer

Address http://aspnet101/webcontrols/calendarsample.aspx

<			February 2002			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	1	2
3	4	5	6	7	8	9

It gets you a lot of functionality with practically no code! That's the type of power web controls give you. They take a given set of functionality and package it up into a nice neat control with methods, properties, and events that make sense based on the type of control involved. For example, the calendar control contains a property SelectedDate that you can use in your code to manipulate the highlighted date and an event SelectionChanged that you can use the perform some action when the selection is changed.

Self Learning Exercise-1

ii. State True or False

a) Web Server controls are just like HTML Controls.

b) ASP.NET requires that all HTML elements must be properly closed and properly nested.

13.3 Custom Control

13.3.1 Developing a Simple Custom Control

It is easy to start authoring your own ASP.NET server controls. To create a simple custom control, all you have to do is to define a class that derives from System.Web.UI.Control and override its Render method. The Render method takes one argument of type System.Web.UI.HtmlTextWriter. The HTML that your control wants to send to the client is passed as a string argument to the Write method of HtmlTextWriter.

13.3.2 Defining Simple Properties

Properties are like "smart" fields that have accessor methods. You should expose properties instead of public fields from your controls because properties allow data hiding, can be versioned, and are supported by visual designers. Properties have get/set accessor methods that set and retrieve properties, and allow additional program logic to be performed if needed.

The following sample shows how to add simple properties that correspond to primitive data types such as integer, Boolean, and string. The sample defines three properties - Message is of type string, MessageSize is of type enumeration, and Iterations is of type integer. Note the page syntax for setting simple and enumeration properties.

13.3.3 Defining Class Properties

If a class A has a property whose type is class B, then the properties of B (if any) are called subproperties of A. The following sample defines a custom control SimpleSubProperty that has a property of type Format. Format is a class that has two primitive properties - Color and Size, which in turn become subproperties of SimpleSubProperty.

Note that ASP.NET has a special syntax for setting subproperties. The following code example shows how to declaratively set the Format.Color and Format.Size subproperties on SimpleSubProperty. The "-" syntax denotes a subproperty.

```
<SimpleControlSamples:SimpleSubProperty Message="Hello There" Format-Color="red"
Format-Size="3" runat=server/>
```

13.3.4 Retrieving Inner Content

Every control has a Controls property that it inherits from System.Web.UI.Control. This is a collection property that denotes the child controls (if any) of a control. If a control is not marked with the ParseChildrenAttribute or marked with ParseChildrenAttribute(ChildrenAsProperties = false), the ASP.NET page framework applies the following parsing logic when the control is used declarartively on a page. If the parser encounters nested controls within the control's tags, it creates instances of them and adds them to the Controls property of the control. Literal text between tags is added as a LiteralControl. Any other nested elements generate a parser error.

The following sample shows a custom control, SimpleInnerContent, that renders text added between its tags by checking if a LiteralControl has been added to its Controls collection. If so, it retrieves the Text property of the LiteralControl, and appends it to its output string.

If your custom control derives from WebControl, it will not have the parsing logic described in the sample, because WebControl is marked with ParseChildrenAttribute(ChildrenAsProperties = true), which results in a different parsing logic. For more information about the ParseChildrenAttribute, see the SDK documentation.

Self Learning Exercise-2

i. Fill in the Blanks

a) Every control has a Controls property that it inherits from ________________.

b) The Render method takes one argument of type _________________.

13.4 Dialog Box

Using dialog box in any application is very important in order to have close talk between the user and the application. These dialogs between the user and the application greatly enhance the usability of the application. As it allow your application to have dialogs with the user more closely.

Dialog box can be used to give some feedback to the user or to get some input from the user or even both.

It is really simple to implement a dialog box in the window based application.

```
Dim fChild As New frmChild
frmChild.Show vbModal
```

It requires some bit technique to open it, specially if you want to open a modal dialog box. In the following article, we will see some of the important technique of implementing dialog box in the web-based application.

13.4.1 Simple Dialog Box

The first and usual technique of implementing dialog box is to put JavaScript to the particular event (usually click event) of the control. It is really an easy way when your application already know when to open a dialog box when the page load to the browser.

Here is the simple example, put a button form control to the web page and insert the following code.

```
btnOpen1.Attributes.Add("onclick", "alert('GOT IT?');")
```

its a one line code to pop a alert message to the user.

Same way, suppose you want that, the application open a confirmation dialog box from where the user can select his option. You can use this method very well. Like, you want to open a confirmation dialog box when user clicks on the delete button of the page. Simply use following code for that:

```
btnDelete.Attributes.Add("onclick", "if(confirm('Are you sure to delete?')){}else{return false}")
```

It will create a small javascript code to the browser. When user selects "No" from the confirmation dialog box, it will return false. Hence nothing will happen. And if user select "Yes", it will return true and will post the page.

13.4.2 Open Dialog Box with Another Web Form

To open an another web page as a dialog box we use,

```
btnOpen2.Attributes.Add("onclick", "window.open ('child.aspx')")
```

btnOpen is again a web form control and when user click on this button, it will open another instance of the browser with the page "child.aspx".

It creates one another instance of the browser, put one more application to the task bar, even the dialog box is modeless. It is possible to open a page in the same browser with modal dialog box. Here is the code for that

```
btnOpen3.Attributes.Add("onclick", "window.showModalDialog('child.aspx',
null,'status:no;dialogWidth:370px;dialogHeight:220px;dialogHide:true;help:no;scroll:no');")
```

It simply creates a small javascript code to the page and attach the click even of the control to that script. If you look at the html source code of the page, then you will find we you want to know/ here is the html code of page that is create with above dialog box popup.

```
<input type="submit" name="btnOpen1" value="Open 1" id="btnOpen1" onclick="alert('GOT
IT?');" style="width:64px;Z-INDEX: 101; LEFT: 8px; POSITION: absolute; TOP: 8px" />
<input type="submit" name="btnDelete" value="Delete" id="btnDelete" onclick="if(confirm('Are
you sure to delete?')){}else{return false}" style="width:64px;Z-INDEX: 102; LEFT: 8px;
POSITION: absolute; TOP: 48px" />
<input type="submit" name="btnOpen2" value="Open 2" id="btnOpen2"
onclick="window.open('child.aspx')" style="width:64px;Z-INDEX: 103; LEFT: 8px; POSI-
TION: absolute; TOP: 88px" />
<input type="submit" name="btnOpen3" value="Open 3" id="btnOpen3"
onclick="window.showModalDialog('child.aspx',null,'status:no;dialogWidth:370px;
dialogHeight:220px;dialogHide:true;help:no;scroll:no');"
 style="width:64px;Z-INDEX: 104; LEFT: 8px; POSITION: absolute; TOP: 128px" />
```

13.4.3 Returning Value from the Dialog Box

Suppose you have opened a dialog box for some purpose and you want that it returns some value to the parent web form.

For returning any thing to the parent window from the child dialog box, javascript provides one attribute of the window object, that is

window.returnValue

to understand its working. We create a small web project with two web forms named:

parent.aspx

child.aspx

Parent web form will have one textbox and a button. Look at the following code of the parent web form.

parent.aspx

```
<body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
    <asp:TextBox id="txtValue" style="Z-INDEX: 101; LEFT: 16px; POSITION: absolute;
TOP: 24px" runat="server"></asp:TextBox>
    <asp:Button id="btnOpen" style="Z-INDEX: 102; LEFT: 176px; POSITION: absolute; TOP:
```

```
24px" runat="server" Text="Open...”></asp:Button>
  </form>
</body>
```

parent.aspx.vb

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  btnOpen.Attributes.Add("onclick", "var strReturn;
strReturn=window.showModalDialog('child2.aspx',null,'status:no;dialogWidth:370px;
dialogHeight:220px;dialogHide:true;help:no;scroll:no');
if (strReturn != null) document.getElementById('txtValue').value=strReturn;")
End Sub
```

And, child web form which will have a textbox and two buttons. Look at the following code of the child page.

child2.aspx

```
<body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
    <asp:TextBox id="txtValue" style="Z-INDEX: 101; LEFT: 16px; POSITION: absolute; TOP:
24px" runat="server"></asp:TextBox>
    <asp:Button id="btnOK" style="Z-INDEX: 103; LEFT: 48px; POSITION: absolute; TOP:
56px" runat="server" Text="Ok" Width="56px"></asp:Button>
    <asp:Button id="btnCancel" style="Z-INDEX: 102; LEFT: 112px; POSITION: absolute;
TOP: 56px" runat="server" Text="Cancel"></asp:Button>
  </form>
</body>
```

child2.aspx.vb

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  btnOK.Attributes.Add("onclick", "window.returnValue =
document.getElementById('txtValue').value; window.close();")
  btnCancel.Attributes.Add("onclick", "window.close();")
End Sub
```

The purpose of this small module is, that when user clicks on the button of the parent form, it will open child.apsx file in the dialog box. In the child form user will enter some value in the textbox and it he click on the Ok button it will return that value to the parent form. With that it will update the textbox of the parent form. And of course, if user clicks on the Cancel button, it will simply close the child dialog box without doing anything.

In all of the above cases, there is nothing to do with the ASP.NET. All of things are actually handle by the javascript. See the javascript code of the child web form.

```
<input name="txtValue" type="text" id="txtValue" style="Z-INDEX: 101; LEFT: 16px; POSI-
TION: absolute; TOP: 24px" />
```

```
<input type="submit" name="btnOK" value="Ok" id="btnOK" onclick="window.returnValue = document.getElementById('txtValue').value; window.close();" style="width:56px;Z-INDEX: 103; LEFT: 48px; POSITION: absolute; TOP: 56px" />
<input type="submit" name="btnCancel" value="Cancel" id="btnCancel" onclick="window.close();" style="Z-INDEX: 102; LEFT: 112px; POSITION: absolute; TOP: 56px" />
```

In the child web form, the value to be return is assigned to the window.returnValue. And the return value is assigned to a variable in the parent web form. Thereafter, it assigned to the textbox.

The idea is that, window.returnValue returns some value from the child window to the parent window, and thereafter, you can do any thing with that returned value. You can even pass to the VB code using parameter of the query string of the document action.

There will be one limitation of using above method of returning value from the dialog box. And the limitation is that you can only return one value to the parent window.

One thing you must have notice that I have used response object for closing the dialog box. Writing javascript script to the response object is another way to put javascript to the web form. In above code it simply create script, which contains window.close() statement, and write it to the browser. At client side of the browser, browser reads the script and executes it.

So, from the last example we learnt new way to put javascript to the browser and that is using response object.

You can use response object to popup a alert message also. Here is the code for that:

```
Private Sub btnOpen5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOpen5.Click
  Dim strScript As String = ""
  strScript = "<script>"
  strScript = strScript & "alert('Simple alter message!!!!');"
  strScript = strScript & "</script>"
  Response.Write(strScript)
End Sub
```

Above code, simply pops an alert message to the user.

If you want to display some variable message in the alert message then it you can do that with slight change.

```
Private Sub btnOpen6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOpen6.Click
  Dim strScript As String = ""
  Dim intSum As Integer
  strScript = "<script>"
  intSum = 344 + 233
  strScript = strScript & "alert('SUM : " & intSum & "');"
  strScript = strScript & "</script>"
  Response.Write(strScript)
```

```
End Sub
```

there will be a drawback again, in using above method to pop alert box, you will notice that if you have tried the above code. We will talk about it later on.

Same thing can be done by registering javascript code to the browser. Following is the example, will do the same thing, but here we will register the javascript to the page.

```
Private Sub btnOpen7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnOpen7.Click
  Dim strScript As String = “”
  strScript = “<script>”
  strScript = strScript & “alert(‘Simple alter message!!!!’);”
  strScript = strScript & “</script>”
  Page.RegisterClientScriptBlock(“ClientScript”, strScript)
End Sub
```

Now about that drawback, yes, you are right, it wash screen of the parent window when it pop the alert message or dialog box. The all controls of the parent window will remain hidden unless you close the alert box or dialog box.

It is because, the javascript that is generated is attached to the top of the html page. You can easily understand this you see the generated html code of the page. Look at the following:

This Code is generated using Response object:

```
<script>alert(‘Simple alter message!!!!’);</script>
<HTML>
<HEAD>
  <title>WebForm1</title>
</HEAD>
<body MS_POSITIONING=”GridLayout”>
</body>
</HTML>
```

Or the Code generated using RegisterClientScriptBlock method :

```
<HTML>
<HEAD>
<title>WebForm1</title>
</HEAD>
<body MS_POSITIONING=”GridLayout”>
  <form name=”Form1" method=”post” action=”MainForm.aspx” id=”Form1">
  <script>alert(‘Simple alter message!!!!’);</script>
    </form>
</body>
</HTML>
```

Self Learning Exercise-3

i. Fill in the Blanks

c) Using __________ in any application is very important in order to have close talk between user and the application.

13.5 Client and Server HTML Controls

In concerning about attribute 'id' is role ends on client-side, i.e., this attribute it's only used by browsers engines and by programmers to, easily, find an html element in page. This very useful attribute is not required, and thus, is only rendered when programmers explicitly set ASP.NET Controls ID property value.

If you heavily use client-side scripting then you, certainly, thought why this attribute isn't always rendered. Well, try thinking on the opposite scenario, where all html elements have its own id attribute rendered, even elements that usually are not changed by script (like labels) and all the other that in your case are not needed. This last scenario seems to me a waste of resources, valuable resources like network bandwidth.

The role of attribute 'name' is far most complex. In fact, most of richness of ASP.NET user experience is built on top of its uniqueness.

Unlike 'id' the 'name' attribute is almost always rendered, and even when is not rendered directly to element attributes it could still be in use on script statements (usually __doPostBack).

The 'name' attribute has two tasks to accomplish:

The first come from the old ASP times and is the fact that 'name' value is used as key in the post values server-side collection (HttpContext.Current.Request.Form)

The second, a new responsibility added by ASP.NET, is to delegate on 'name' value the responsibility of identifying uniquely the element source of current post back.

This last responsibility adds a new constrain to 'name' attribute value: it must be unique but it also must be something that maps directly to a single Server-Side Control, the same that previously render the 'name' attribute value.

To achieve this task, every ASP.NET Control has a UniqueID property that is built based on the control hierarchy. In order to ensure that a control always has the same UniqueID its assumed that his parent hierarchy is always built the same way, even dynamic controls should always be added at the same point of the Page Life Cycle.

the bridge between client and server-side that allows the raise of server-side control events like OnClick and OnChange (OnTextChange, OnSelectIndexChange, etc.).

13.6 HTML Server Controls

13.6.1 HTML tags

HTML server controls are HTML tags understood by the server.

HTML elements in ASP.NET files are, by default, treated as text. To make these elements programmable, add a runat="server" attribute to the HTML element. This attribute indicates that the element should be treated as a server control.

All HTML server controls must be within a <form> tag with the runat="server" attribute.

ASP.NET requires that all HTML elements must be properly closed and properly nested.

HTML Server Control	**Description**
HtmlAnchor	Controls an <a> HTML element
HtmlButton	Controls a <button> HTML element
HtmlForm	Controls a <form> HTML element
HtmlGeneric	Controls other HTML element not specified by a specific HTML server control, like <body>, <div>, <span>, etc.
HtmlImage	Controls an <image> HTML element
HtmlInputButton	Controls <input type=”button”>, <input type=”submit”>, and <input type=”reset”> HTML elements
HtmlInputCheckBox	Controls an <input type=”checkbox”> HTML element
HtmlInputFile	Controls an <input type=”file”> HTML element
HtmlInputHidden	Controls an <input type=”hidden”> HTML element
HtmlInputImage	Controls an <input type=”image”> HTML element
HtmlInputRadioButton	Controls an <input type=”radio”> HTML element
HtmlInputText	Controls <input type=”text”> and <input type=”password”> HTML elements
HtmlSelect	Controls a <select> HTML element
HtmlTable	Controls a <table> HTML element
HtmlTableCell	Controls <td>and <th> HTML elements
HtmlTableRow	Controls a <tr> HTML element
HtmlTextArea	Controls a <textarea> HTML element

Web server Control

Web server controls are special ASP.NET tags understood by the server.

Web Server Controls

Like HTML server controls, Web server controls are also created on the server and they require a runat=”server” attribute to work. However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.

The syntax for creating a Web server control is:

<asp:control_name id=”some_id” runat=”server” />

Web ServerControl	**Description**
AdRotator	Displays a sequence of images
Button	Displays a push button
Calendar	Displays a calendar
CheckBox	Displays a check box
CheckBoxList	Creates a multi-selection check box group
DataGrid	Displays fields of a data source in a grid
DataList	Displays items from a data source by using templates

DropDownList	Creates a drop-down list
HyperLink	Creates a hyperlink
Image	Displays an image
ImageButton	Displays a clickable image
Label	Displays static content which is programmable (lets you apply styles to its content)
LinkButton	Creates a hyperlink button
ListBox	Creates a single- or multi-selection drop-down list
Literal	Displays static content which is programmable (does not let you apply styles to its content)
Panel	Provides a container for other controls
PlaceHolder	Reserves space for controls added by code
RadioButton	Creates a radio button
RadioButtonList	Creates a group of radio buttons
Repeater	Displays a repeated list of items bound to the control
Table	Creates a table
TableCell	Creates a table cell
TableRow	Creates a table row
TextBox	Creates a text box
Xml	Displays an XML file or the results of an XSL transform

Self Learning Exercise-4

i. Fill in the Blanks

d) HTML Server Controls are _________ tags understood by the server.

e) All HTML server controls must be within a ________ tag with the runat=”server” attribute.

f) ___________ control creates a hyperlink button.

ii. State True or False

c) AdRotator Control displays the sequences of rotation in a text.

13.7 Windows Services

Windows services are applications that run outside of any particular user context in Windows NT, Windows 2000, or Windows XP. The creation of services used to require expert coding skills. You can also write a Windows service in any other language that targets the common language runtime. This article walks you through the creation of a useful Windows service, then demonstrates how to install, test, and debug the service.

If you’re running Windows NT® or Windows® 2000, you’re running at least a handful of Windows services (previously called Windows NT Services). Typically these services provide system-level support, including the system event log, task scheduler, and telephony. An important aspect of Windows services is that they can run without a user context, albeit only under Windows NT, Windows 2000, or Windows XP at this point. They don’t require a user to be logged in order to do their work, and they generally run in a higher-powered security mode than do most users.

Windows services are often started by Windows when you boot up, but can be stopped, paused, restarted, and shut down from the Services applet provided by Windows. In addition, you can manage Windows services programmatically, and Microsoft® .NET makes this easy, using the ServiceController component that's part of the .NET Framework.

Creating a Windows service in Visual Studio .NET is amazingly simple. If you follow the steps, you end up with a service, and debugging the service is much simpler than in previous versions.

This article focuses on creating a simple, somewhat useful Windows service which monitors file changes on a specific drive, or, with modification, on multiple drives. As you follow the steps to build this Windows service, I'll explain the EventLog and FileSystemWatcher components provided by the Microsoft .NET Framework. You'll see how to add an installer to your project, so that the InstallUtil program that comes with Visual Studio can install your service, and you'll see how to debug your service as well, taking advantage of the ServiceController component that is provided by the .NET Framework.

13.7.1 Creating a Windows Service in .NET

From a developer's perspective, a Windows service application exists as an executable file, although this file may contain more than one service. You determine the behavior of each service in your project by writing a class for each that inherits from the ServiceProcess.ServiceBase class (creating a Windows Service project takes care of most of these types of details for you), and adding code to handle the various methods provided by this class. Among other things, the service you create can provide code for the OnStart, OnStop, OnPause, OnContinue, and OnShutdown methods called by the Service Control Manager (SCM) as you interact with the service. None of these procedures is required, but you can use them to provide specific behavior in reaction to requests from the SCM's user interface, or from other services. (The Service Control Manager mentioned here is, by the way, not the same as the service COM uses to bootstrap objects, even though they share the same name.)

The service class you create can't install itself, however. You must also supply an Installer class that inherits from Configuration.Install.Installer. For each service in your project, this class creates one ServiceProcessInstaller object (which knows how to install the service with the SCM for you), and a ServiceInstaller object. The ServiceInstaller object writes information to your registry for you as needed for the installation—specifically, to a subkey under the HKEY_LOCAL_MACHINE\ System \CurrentControlSet \Services key.

Once you've created your service class with the appropriate functionality built in, you'll follow these steps to get it up and running:

Compile your project to create an EXE.

Use the InstallUtil program that's part of the .NET platform to install your project (you should load it as a service).

Use the Services applet from the Control Panel to start your service. By default, the project you created sets the start type to Manual.

If you want to debug your service, now's the time to grab onto the process containing your service, and step through the code looking for problems.

This article covers each of these steps in some detail.

Walkthrough

To illustrate the process, we will be creating a simple service that actually does something useful rather than using the example provided by Microsoft which writes "Hello, World" to your event log every second it runs. The service monitors file activity on the drive that you specify when you start the service. This service writes informational messages into your application's event log, indicating that you've changed an item in the file system underneath the path you specify.

The .NET platform, provides the FileSystemWatcher component, which makes it easy to determine when users have modified, deleted, added, or moved files on a drive. All you have to do is place one of these components on the design surface for your service class, and you can react to events raised by the component when files are changed. It's even easier to take advantage of event logs—the EventLog class is provided by the .NET Framework as well, so you can just drop it and use it.

To create this, follow these steps:

Start Visual Studio .NET, and create a new project. From the New Project dialog box, choose either a Visual Basic or Visual C#™ project, and select the Windows Service project type. Set the name for the project to FileWatcher, and place it in a convenient folder. (When entering code throughout this article, choose the appropriate code sample, C# or Visual Basic .NET, matching your project type.

You don't have to use the Windows Service project template, but if you choose not to, you'll need to do a lot more work on your own. Selecting the project template sets up the appropriate inheritance, and includes the necessary components.

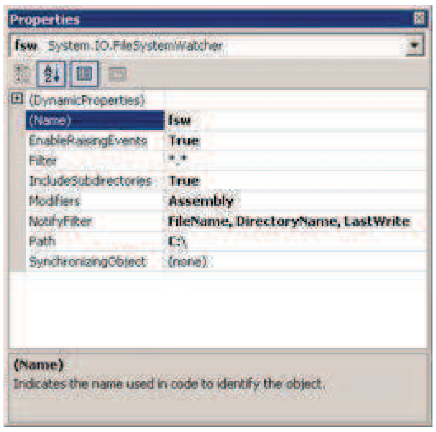
Modify the file name. Once Visual Studio has created the project, you'll find a class named Service1 within the project. In the Solution Explorer window, rename this file FileWatcher.cs or FileWatcher.vb, depending on the project language.

Open the class in Design view. With the FileWatcher class selected, you can either press Shift+F7 or choose the View | Designer menu item.

Modify the service properties. In the Properties window, set the service's properties. The three Boolean properties (CanPauseAndContinue, CanShutDown, and CanStop) control the behavior of the running service. The ServiceName property provides the name the system uses to refer to the service. (You might also investigate the CanHandlePowerEvent, which indicates that the service handles the computer power status changes indicated in the PowerBroadcastStatus class. Modifying that property isn't really necessary in this example.)

Add the FileSystemWatcher component. With the FileWatcher class still open in Design view, locate the Toolbox window (press Ctrl+Alt+X, or use the View | Toolbox menu item to ensure that it's visible). Find the Components tab, and then double-click on the FileSystemWatcher component to add an instance to the designer. Modify the component's name to be fsw.

Set fsw's properties. Select the FileSystemWatcher component you just added and modify its properties. Although you can accept the FileSystemWatcher object's default properties, you'll probably always want to set at least the Filter, Path, and IncludeSubdirectories properties. The following **figure** shows the FileSystemWatcher object's Properties window, once you've set all its properties.



Properties

fsw System.IO.FileSystemWatcher

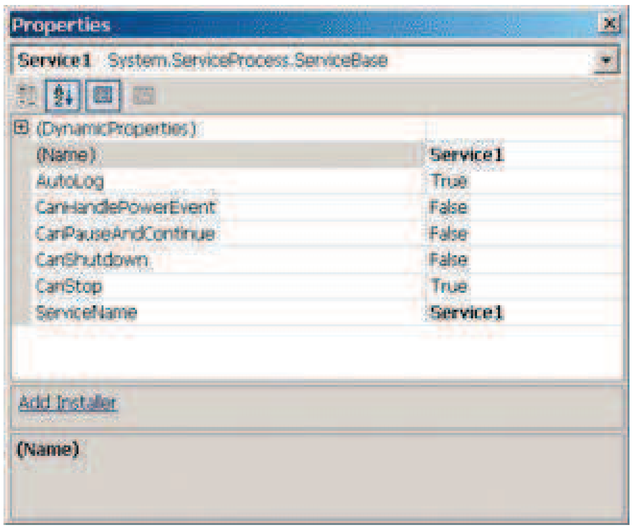
(DynamicProperties)	
(Name)	fsw
EnableRaisingEvents	True
Filter	*.*
IncludeSubdirectories	True
Modifiers	Assembly
NotifyFilter	FileName, DirectoryName, LastWrite
Path	C:\
SynchronizingObject	(none)

(Name)
Indicates the name used in code to identify the object.

Add startup code. Generally, you'll need code in your service's OnStart event that sets up your service and its data structures, and perhaps logs information to the system event log. In this case, your service will simply set fsw's Path property based on the arguments passed into the service, and will log the event to the event log. Use the View | Code menu item (or press F7) to load the class's module, find the existing OnStart procedure, and modify it

You can also write code to handle the OnContinue, OnCustomCommand, OnPause, OnShutdown, OnPowerEvent, and OnStop events of the service.

Add an Installer. Select the View | Designer menu item (or press Shift+F7). Make sure the Properties window is visible (use the View | Properties window menu item, or press F4), and click on the designer surface to make sure it's selected. Then select the Add Installer link at the bottom of the Properties window to add the installer classes.



Properties

Service1 System.ServiceProcess.ServiceBase

(DynamicProperties)	
(Name)	Service1
AutoLog	True
CanHandlePowerEvent	False
CanPauseAndContinue	False
CanShutdown	False
CanStop	True
ServiceName	Service1

Add Installer

(Name)

13.7.2 Investigating Installers

Once you've selected the Add Installer link, Visual Studio .NET creates a new project file named ProjectInstaller.cs (or .vb). Right now, this new file contains two components: ServiceProcessInstaller1 and ServiceInstaller1. If you investigate the properties of each, you'll see that the ServiceInstaller1 object includes a ServiceName property. This component installs your particular service. The other component, ServiceProcessInstaller1, has properties such as Account, Password, and UserName. This component takes care of interacting with the Windows SCM on behalf of your service. You may have more than one ServiceInstaller component in your project—if your project contains multiple classes that inherit from ServiceBase, for example—but it will only have a single instance of a ServiceProcessInstaller object.

Note: that Visual Studio .NET sets the ServiceName property of the ServiceInstaller object when you add the ServiceInstaller object to your project. If you change the name of your service class, you'll need to manually change this property to match.

There are a few important things that you should note. The class that Visual Studio .NET creates for you, inheriting from System.Configuration.Install.Installer (named ProjectInstaller by default), must include the RunInstaller attribute, set to True, in order for the installation to be invoked when you install the assembly. If you remove this or set it to False, you service won't be installed. Like other classes that include a Designer window, this one provides an InitializeComponent method. The intent is that you won't modify this—make all your changes in the designer. Code you might add, however, would be in reaction to the various events raised by the ServiceProcessInstaller or ServiceInstaller classes, including Before/AfterCommit, Before/AfterInstall, Before/AfterRollback, and Before/AfterUninstall. These events give you a chance to modify the default behavior for the installer classes that you add to your projects.

By default, the ServiceInstaller class's StartType property is set to Manual. If you want your service to start automatically when you install it, select the ServiceInstaller1 component in ProjectInstaller's designer, and set the StartType property to Automatic.

In order for the installer to be able to install your service, you need to supply information about the account the service will run as. For this simple service, you can set the Account property of ServiceProcessInstaller1 to be LocalSystem—that way, you needn't supply any authentication information. However, if you don't set the Account property to LocalSystem, or you don't supply a valid UserName and Password property values, the installation setup will fail.

13.7.3 Adding FileSystemWatcher Event Code

The FileSystemWatcher component provides a number of events to which you can react in code, including Changed, Created, Deleted, and Renamed.

The designer will take you to the appropriate event handler stub, and will add the necessary code to hook up the event, as well. Follow these steps to set up your code:

Select the FileWatcher object in the Solution Explorer window, and then use View | Designer (Shift+F7) to open the designer window.

Select the FileSystemWatcher component on the FileWatcher class designer.

Add code for the Changed event. In reaction to the Changed event, you'll write the type of change, plus the name of the file or folder, to the event log. In the code window, select fsw from the left dropdown list at the top of the window, and then select Changed from the right-hand list.

Add code for the Created event. In reaction to the Created event, write the name of the new file or path to the event log.

Add code for the Deleted event. In reaction to the Deleted event, write the name of the deleted file or path to the event log

Add code for the Renamed event.

If you're creating a Visual Basic .NET project, you're all finished. If you're creating a C# project, however, you'll need to manually hook up the event handlers for the event procedures you just added. (Visual Basic .NET provides this internal plumbing for you—C# does not.) Therefore, if you're creating a C# project, follow this final step: in the FileWatcher.cs module, find the InitializeComponent procedure. Ensure that the lines of code appear before the call to the EndInit method of the fsw object. These lines add event handlers for the procedures you've created.

If you used the Events button on the Properties windows toolbar to create the event procedures, you won't need to add the code in the final step yourself—Visual Studio will have added it for you. If you're using Visual Basic .NET, you needn't worry about these event handlers, since Visual Basic .NET takes care of the hookups under the covers.

13.7.4 Testing Service

Unlike many applications you'll create using Visual Studio .NET, you cannot simply run a Windows service from within the development environment. You must build the executable, install it, and test it as a running service. (Debugging a service is another story altogether, which I'll cover in the next section.) Follow these steps to test your Windows service:

Use the File | Save All menu item, or press Ctrl+Shift+S, to save your entire project.

Choose the Build | Build menu item (or press Ctrl+Shift+B) to build your service's executable file.

Find the executable. Use the Visual Studio .NET Command Prompt item from the Start menu. It's installed as a subitem of the Visual Studio .NET Tools item to open a Windows command prompt. (If you don't follow these steps, you won't have your MS-DOS path set correctly.) Change to the folder where your project is stored. Navigate to the bin\debug folder for C# projects, or the bin folder for Visual Basic projects, and locate the FileWatcher.exe file you've just created.

Install your service. Run the InstallUtil tool that comes with Visual Studio, using the following command line:

```
InstallUtil filewatcher.exe
```

Start the service. Unless you modified the ServiceInstaller1 StartType property, you'll need to start your service. Bring up the Windows service manager. (The exact steps depend on your operating system. In Windows 2000, you can use the Administrative Tools applet in Control Panel and select the Services item.) In the Services tool, find the FileWatcher service and double-click it to load the properties dialog box for the service. If you want to specify a path other than C:\ for the FileWatcher class, you can do that in the Start Parameters textbox. When you're ready, press Start to start the service.

Investigate the Startup events. Open the Event Viewer (again, the steps vary depending on your operating system). Select the Application log, and note that you should find two entries in the event log already. The first, sent from your service's OnStart event procedure, contains the text "FileWatchService starting. There are 0 args. Watch path is 'C:\'." (The text will be different if you specified an argument when you started the service.) The second, sent automatically (because you set the service's AutoLog property to True), indicates that the service started successfully.

Test the service. Open Windows Explorer, navigate to the folder your service is "watching," and make some changes. Rename, delete, or modify files. Switch back to the event viewer, and you should see information about your changes logged there.

Remove the service. When you're done, use the Services dialog box to stop your service. Run InstallUtil with its /u parameter to uninstall your service. You'll also need to shut down the Windows Services applet to completely remove your service from memory.

13.7.5 Debugging Service

At some point, you're likely to want to single-step through the code you've written in your service application. Although you can debug services much as you debug other applications, you cannot simply press F5 to start running the service from within the Visual Studio development environment, as you can with other projects.

Because you must compile and install a service before running it, debugging becomes a little more complex than for normal applications. Visual Studio .NET makes this process as easy as possible. Once your service is running, you can grab onto the running process and debug your code. To test this out, follow these steps:

Reinstall and start your service, as described in the previous section. (You must start the service running, or you won't be able to debug it.)

In Visual Studio, with your project loaded, select the Debug | Processes menu item, displaying the Processes dialog box.

Select the Show system processes checkbox so that the Available Processes list includes running services.

In the Available Processes list, select the Filewatcher.exe entry, and then click Attach. On the Attach to Process dialog box, make sure that the Common Language Runtime option is selected, then select OK to accept. Then you can close the Processes dialog box.

Set a breakpoint in your code at the location you'd like to test. In this case, set a breakpoint on one of the event procedures you've created within the FileWatcher class.

Trigger the breakpoint by taking the necessary action within the file system. For example, if you set a breakpoint in the Created event procedure, creating a new folder in the "watched" folder should bring you back to your breakpoint. At this point, you can single-step through the code, as you would with any other application. Press F5 to continue running when you're finished with your testing.

When you're finished debugging, detach the debugger from the running process using the Debug | Processes menu. Select the FileWatcher project in the list of debugged processes, then click the Detach button that's to the right of the list. Dismiss the dialog box.

Use the Debug | Stop Debugging menu item to end the debugging session.

One solution is to cheat—that is, insert a pause into the OnStart procedure, using code like this

```
System.Threading.Thread.Sleep(25000);
```

This code causes the thread to sleep for 25 seconds. This is long enough for you to dig through the Debug | Processes menu item, find the process you want to debug, and attach to it, but not so long that the SCM times out. Once you've added this code, you can step through your OnStart procedure.

Self Learning Exercise-5

i. Fill in the Blanks

g) ______________ are applications that run outside of any particular user context in Windows NT, Windows 2000 or Windows XP.

ii. State True or False

d) Windows Services are often started by windows when you boot up, but can not be stopped, paused, restarted and shutdown.

13.8 Web Services

Web Services promise to bring information into your applications from the Internet in much the same way that browser have made information available to end users. The .Net framework introduces Web Services as an integral part of the architecture, making it very easy to create and consume these services with minimal amounts of code written. In fact, if you read Microsoft's documentation, Web Services are featured as the new component architecture in the distributed age where not only Internet exposure is handled through them but also common reusable business and application services.

The .Net framework abstracts most of the internal logic that handles the remoting details of method calls over the wire and Visual Studio .Net builds support for Web Services directly into the development environment. With all of this in place it becomes almost as easy to call a remote method as it is to call a local method. And that after all is what Web Services are about – making server side logic easily available to client applications.

Web Service's mission is to provide a Remote Procedure Call (RPC) interface for client applications to call class methods on the server side. Actually, the handling interface on the server need not be a class, but in the case of .Net and COM before it classes are typically used as the implementing entity. The idea is that in order to create a Web Service, you create a class method with standard input and output parameters and you then mark those classes and the specific methods as exposable over the Net.

Web Services are meant to expose functionality of a server to other applications. The 'client' applications in this case may be a Fat Client Windows app, a Fat Server Web application that runs a standard Web backend such as ASP, Cold Fusion, Web Connection etc., a browser based client application using script code, or even Java applet running in a browser on a Unix machine. As long as a client application has support for the Simple Object Access Protocol (SOAP) it can call the remote Web Service and return data for it, assuming the user is authorized.

SOAP is an important part of this process – it's the protocol that's responsible for routing the RPC message from the client to the server and returning the result back to the client application. SOAP is based on XML and follows a relatively simple design that's easy to implement. SOAP's simple protocol has contributed to its widespread support on just about any platform and development environment. You can find SOAP clients for COM (the MSSOAP Toolkit is available for Visual Studio 6 developers), .Net (obviously), Perl, Java, C++, PHP – and just about any development environment you can think of.

SOAP implementations provided by vendors typically consist of two pieces: A client side Proxy that handles the SOAP message creation and result message cracking to return the result data, as well as a server piece that implements the Web Service logic. The server piece tends to be an application server that calls out to custom Web Service classes that you create and that contain the business logic of your Web Service. The server code you write essentially consists of simple methods to handle inputs and outputs via parameters and return values respectively. The logic you write in the actual method is up to you and contains any functionality that your language of choice supports. This means writing code to call your business objects or if the process is simple enough using procedural code to perform some operation. Although Web Services can expose classes, you'll find that typically you end up creating wrapper classes for existing business objects in order to handle the specific logic required to drive your Web Service. As such you're breaking up the business tier with a front end service (the Web Service) and a business service (your actual business objects – or if you use procedural code just that code).

One important thing to remember is that Web Services follow typical Web rules. For one they are stateless. This means that even though Web Services expose classes, they are more of a remote procedure call interface than a remote class interface. You call methods with parameters rather than storing state in properties between method calls. In fact, none of the major Web Service implementations support properties in any way. If you need to keep state you'll have to use Web Server specific functionality such as the ASP or ASP.NET Session object to store that state and retrieve it on subsequent hits. Since Web Services use the standard Web architecture, the same tools you've used for HTML based browser applications can also be used in the Web Services code, although you will find that you spend very little time accessing this functionality in your Web Service code as the frameworks abstract away the need to deal with the HTTP and Web Server layer for the most part. Things that you may need to manage yourself include state and security since the Web Service architecture doesn't provide this for you. In both cases you can however take advantage of the HTTP services (Authentication, SSL for example) or the Web server features like the session object to make short work of dealing with these issues.

SOAP proxies vary widely in quality and ease of use and unfortunately, as the official SOAP spec is still under heavy construction by the standards bodies. Interoperability is not perfect, but getting better all the time. Be sure to stay up with your vendor's latest toolkits for best compatibility. Proxies are the key to making Web Services easy to use and consume. Microsoft's implementations in both COM and .Net provide proxy interfaces that simulate the remote object and provide you with a simple call interface that lets you create the proxy and then call methods on it the same way as you would on a local object. Visual Studio .Net takes this one step further by actually providing you a real proxy object that contains the actual methods of the remote object making it possible to even use time saving features like IntelliSense on the object.

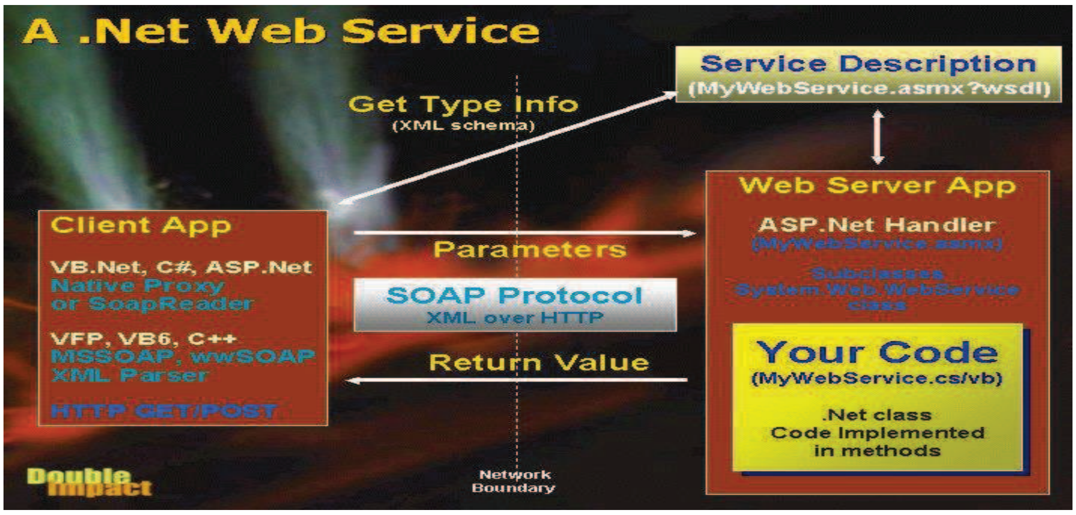
Creating Web Services in .Net and consuming the service either in a Windows Form application or an ASP.Net Web page is almost trivial.For the remainder of this article I'll walk you through creating a Web Service and then creating two clients – one as a Windows Form and one as ASP.Net Web Form to consume that Web service. In this example you will study how Web Services are implemented and how they behave rather than building an elaborate sample application. Before this lets talk about how .Net implements Web Services so you can get an idea of what happens behind all the fancy black box code that .Net provides to make Web Services so easy to use.

Behind the scenes there are three major components that make up a Web Service:

1. The Web Service on the Server side
2. The client application calling the Web Service via a Web Reference
3. A WSDL Web Service description that describes the functionality of the Web Service

Net Web Services use WSDL files to get a type description of the Web Service which provides the detail needed to the client to create a proxy. The proxy calls the Web Service using the SOAP protocol passing parameters and returning a return value for the remote method call.

A Web Service in .Net consists of a .ASMX page that either contains a class that provides the Web Service functionality or references a specific external class that handles the logic in an external class file. Classes are standard .Net classes and the only difference is that every method that you want to expose to the Web is prefixed with a [WebMethod] attribute. Once the .ASMX page has been created the Web Service is ready for accessing over the Web. .Net provides a very useful information page about Web Service showing all the methods and parameters along with information on how to access the Web Service over the Web. You can also use this page to test basic operation of Web Service without calling the Web Service with a 'real' client.



[source:www.programmersheaven.com]

Net Web Services that run over HTTP can be called in 3 different ways:

13.8.1 HTTP GET Operation

You can pass parameters to a Web Service by calling the ASMX page with query string parameters for the method to call and the values of simple parameters to pass.

13.8.2 HTTP POST Operation

Works the same as GET Operation except that the parameters are passed as standard URL encoded form variables. If you use a client such as wwIPStuff you can use AddPostKey() to add each parameter in the proper parameter order.

13.8.3 SOAP

This is the proper way to call a Web Service in .Net and it's also the way that .Net uses internally to call Web Services.

The GET and POST operations are useful if you need to call a Web Service quickly and no SOAP client is readily available. For example, in a browser based client application it may be easier to use GET and POST instead of constructing and parsing the more complex SOAP headers that are passed back and forth in a SOAP request. But with a proper SOAP client in place SOAP provides the full flexibility of the protocol, where GET and POST operations have to stick to simple inputs and outputs. Among other things that you can do with SOAP is pass complex objects and data over the wire and for these operations to work you need to use SOAP.

13.8.4 WSDL – a type library for a Web Service

When you create a Web Service you automatically get support for a WSDL (Web Service Description Language) schema that describes the Web Service by accessing the .ASMX page with a querystring of WSDL:

A WSDL file describes all the methods and method signatures, as well as the namespaces and the handling URL for the Web Service in an XML document. This document works very much like a type library does in COM for the client application to determine what functionality is available in the Web Service. Visual Studio.Net uses the WSDL file to create a Web Reference on the client side from your Web Service. It reads the WSDL file and based on the definitions found in the WSDL file creates a proxy class that mimics the interface of the Web Service. The resulting class is actual source code that you can look at (see Web References sidebar). Because this class is actually linked into your client project the class becomes available in IntelliSense and you can actually see the full interface of the class as you type.

Self Learning Exercise-6

i. Fill in the Blanks

h) Web Service's mission is to provide a ________________ interface for client applications to call class methods on the server side.

i) The full form if WSDL is __________________.

ii. State True or False

e) Web Services promise to bring information into your application from the Internet in much the same way that browsers have made information available to end users.

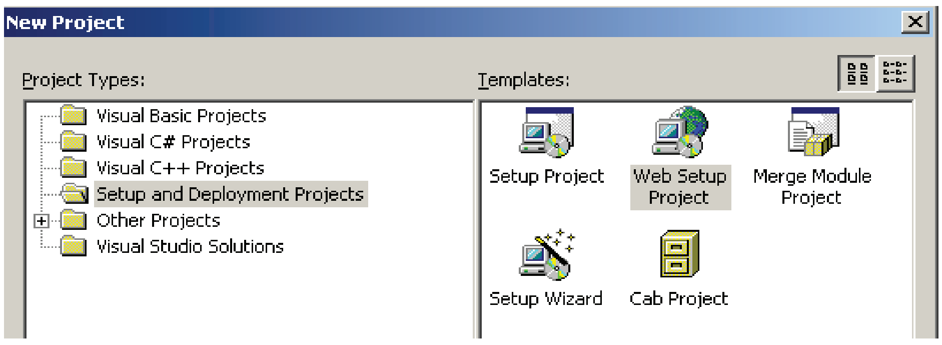
f) SOAP is a protocol that is responsible for routing the RPC message from the client to the server.

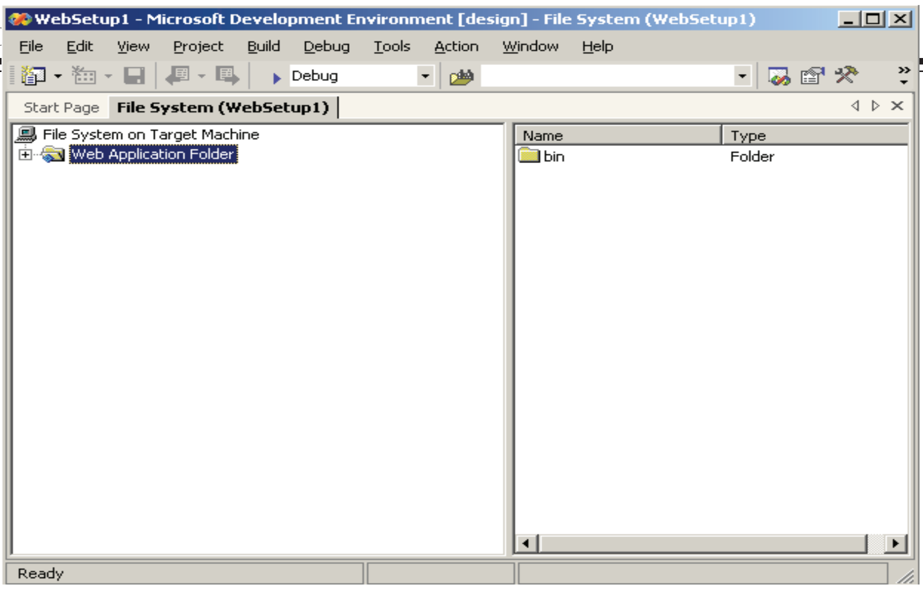
13.9 Deploying ASP.NET Applications

After creating and testing your ASP.NET application, the next step is to deploy the application. Deployment is the process of distributing the finished application to be installed on other computer. We can use the built-in deployment feature that comes with Visual studio .NET to create a Windows Installer file - a .msi file for the purpose of deploying applications.

13.9.1 Deploying Applications

To start, open the Web Application project you want to deploy. Say, you have a project named "Deploy" with ten Web pages in it. Select File->Add Project->New Project from the main menu. From the Project Types pane select Setup and Deployment Projects and from the Templates pane select Web Setup Project. Type WebSetup1 for name and specify a location in the location box and click OK. The New project dialogue box looks like the image below.

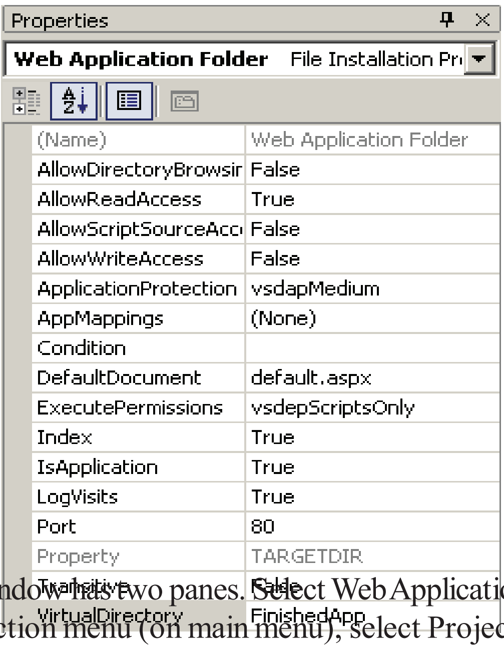



WebSetup1 - Microsoft Development Environment [design] - File System (WebSetup1)
File Edit View Project Build Debug Tools Action Window Help
Debug
Start Page
File System (WebSetup1)
Ready

File System on Target Machine
Web Application Folder

Name	Type
bin	Folder

When you click OK on the above dialogue box, the project is added to the solution explorer window and also a File System Editor window appears as shown in the figure.

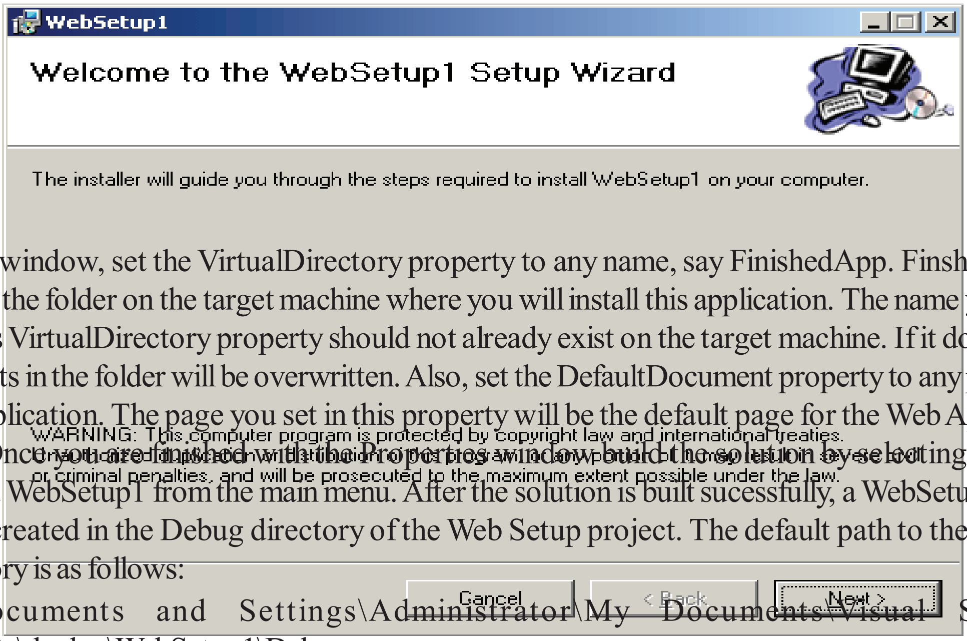


(Name)	Web Application Folder
AllowDirectoryBrowsir	False
AllowReadAccess	True
AllowScriptSourceAcc	False
AllowWriteAccess	False
ApplicationProtection	vsdapMedium
AppMappings	(None)
Condition	
DefaultDocument	default.aspx
ExecutePermissions	vsdepScriptsOnly
Index	True
IsApplication	True
LogVisits	True
Port	80
Property	TARGETDIR
Transitive	False
VirtualDirectory	FinishedApp

The File System Editor window has two panes. Select Web Application Folder in the left pane in this window. From the Action menu (on main menu), select Project Output to open the Add Project Output Group dialog box. It looks like the image below.

Make sure that Deploy is selected in the Project drop-down list and select Primary Output from the list and click OK. You also can select other options depending upon the users of your application.

Now, in the File System Editor window, select Web Application Folder and open it's Properties window. The Properties window for the Web Application Folder looks like the image below.



Welcome to the WebSetup1 Setup Wizard

The installer will guide you through the steps required to install WebSetup1 on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

You can use the default Virtual Directory specified by the installer or you can specify one. Click next to install the application.

In this window, set the VirtualDirectory property to any name, say FinishedApp. FinshedApp will be the folder on the target machine where you will install this application. The name you set for this VirtualDirectory property should not already exist on the target machine. If it does, the contents in the folder will be overwritten. Also, set the DefaultDocument property to any page in the application. The page you set in this property will be the default page for the Web Application. Once you are finished with the Properties window build the solution by selecting Build->Build WebSetup1 from the main menu. After the solution is built sucessfully, a WebSetup1.msi file is created in the Debug directory of the Web Setup project. The default path to the debug directory is as follows:

C:\Documents and Settings\Administrator\My Documents\Visual Studio Projects\deploy\WebSetup1\Debug

13.9.2 Installing the Application

You can copy the WebSetup1.msi file to the target machine and double-click to install the Web Application. When you double-click the setup file the dialog that opens looks like the image below.

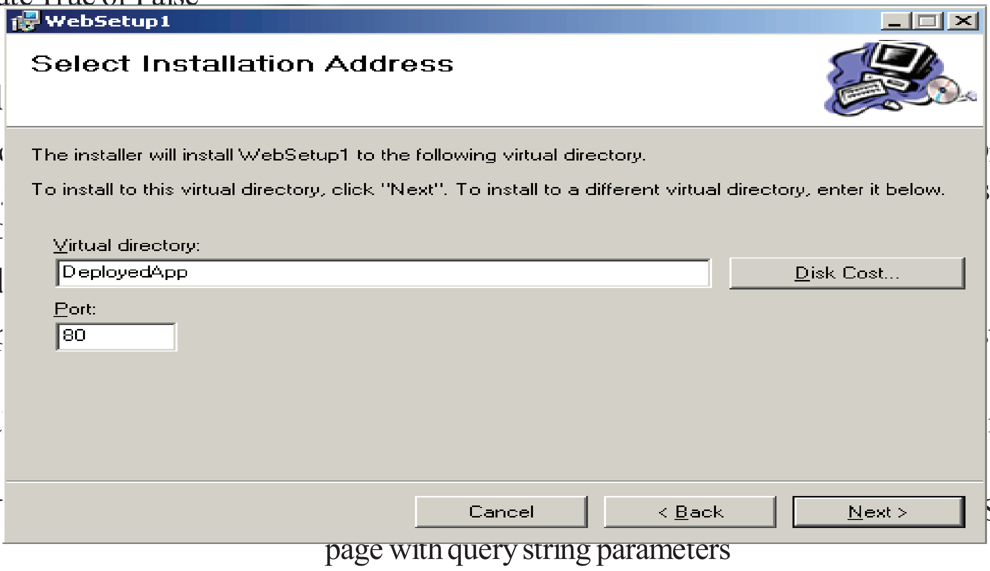
Self Learning Exercise-7

i. Fill in the Blanks

j) ____________ is the process of distributing the finished application to be installed on other computer.

Click next and you will be taken to the next dialog which looks like the image below.

ii. State True or False



Select Installation Address

The installer will install WebSetup1 to the following virtual directory.

To install to this virtual directory, click "Next". To install to a different virtual directory, enter it below.

Virtual directory: DeployedApp

Port: 80

13.1

In sh ifferent dialo for the deplo

13.1

Dep	talled
Dia	input
GE	SMX page with query string parameters
POST	Parameters are passed as standard URL encoded form variables.
SOAP	Simple Object Access Protocol , the protocol that's responsible for routing the RPC message from the client to the server and returning the result back to the client application
Web Services	Provides a Remote Procedure Call (RPC) interface for client applications to call class methods on the server side.
Windows Services	Applications that run outside of any particular user context in Windows NT, Windows 2000, or Windows XP.
WSDL	Web Service Description Language, describes all the methods and method signatures

13.12 Further Readings

1. Mridula Parihar, Essam Ahmed, Jim Chandler, Bill Hatfield, "ASP.NET Bible", Unique Color carton, New Delhi

2. Mathew Macdonald, "ASP.NET: The Complete Reference", Tata Mcgraw Hill, New Delhi

3. John Alexander, Billy Hollis, "Developing Web Applications with Visual Basic .NET and ASP.NET", John Wiley New Delhi

13.13 Answers to Self Learning Exercises

i. Fill in the Blanks	ii. True/False
System.Web.UI.Control	a) False
System.Web.UI.HtmlTextWriter	b) True
Dialog Box	c) False
HTML	d) False
<form>	e) True
LinkButton	f) True
Windows Services	g) True

Unit - 14: File Handling in VB.NET

Structure of the Unit

14.0 Objective

After going through this unit you will understand,

- features of file handling
- multithreading in .NET
- how to catch the run time error
- how to write an error handling routine
- how to fetch the data from database
- how to display the data on your web page using Data Grid , Repeater, and Data List control.

14.1 Introduction

From the previous Unit you have learned about how to make a web pages in .NET. You have also learned various controls that can be placed on a web page.

In this Unit we will discuss about file handling, multithreading and DataBase Connectivity

Before going through this unit it is required that you must read Unit-12 & Unit-13 and solve the exercises.

14.2 File Handling

File handling in Visual Basic is based on System.IO namespace with a class library that supports string, character and file manipulation. These classes contain properties, methods and events for creating, copying, moving, and deleting files. Since both strings and numeric data types are supported, they also allow us to incorporate data types in files. The most commonly used classes are FileStream, BinaryReader, BinaryWriter, StreamReader and StreamWriter.

14.2.1 FileStream Class

This class provides access to standard input and output files. We use the members of FileAccess, FileMode and FileShare Enumerations with the constructors of this class to create or open a file. After a file is opened it's FileStream object can be passed to the Binary Reader, BinaryWriter, Streamreader and StreamWriter classes to work with the data in the file. We can also use the FileStreamSeek method to move to various locations in a file which allows to break a file into records each of the same length.

14.2.2 StreamReader and StreamWriter Class

The StreamReader and StreamWriter classes enables us to read or write a sequential stream of characters to or from a file.

14.2.3 BinaryReader and BinaryWriter Class

The BinaryReader and BinaryWriter classes enable us to read and write binary data, raw 0's and 1's, the form in which data is stored on the computer.

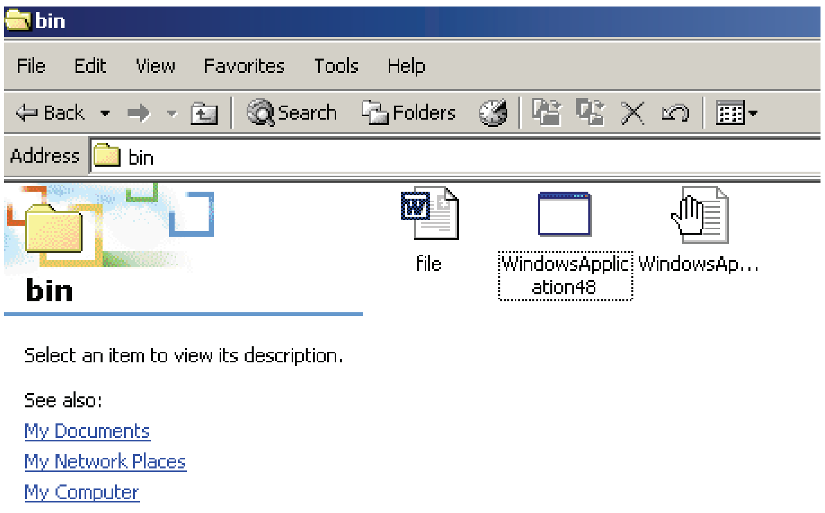
The following examples puts some code to work with textual data using FileStream and StreamReader and StreamWriter classes.

14.2.4 Code to create a File

```
Imports System.IO
‘NameSpace required to be imported to work with files
Public Class Form1 Inherits System.Windows.Forms.Form
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)
'declaring a FileStream and creating a word document file named file with
'access mode of writing
Dim s as new StreamWriter(fs)
'creating a new StreamWriter and passing the filestream object fs as argument
s.BaseStream.Seek(0,SeekOrigin.End)
'the seek method is used to move the cursor to next position to avoid text to be
'overwritten
s.WriteLine("This is an example of using file handling concepts in VB .NET.")
s.WriteLine("This concept is interesting.")
'writing text to the newly created file
s.Close()
'closing the file
End Sub
End Class
```

The default location where the files we create are saved is the bin directory of the Windows Application with which we are working. The following figure displays that.



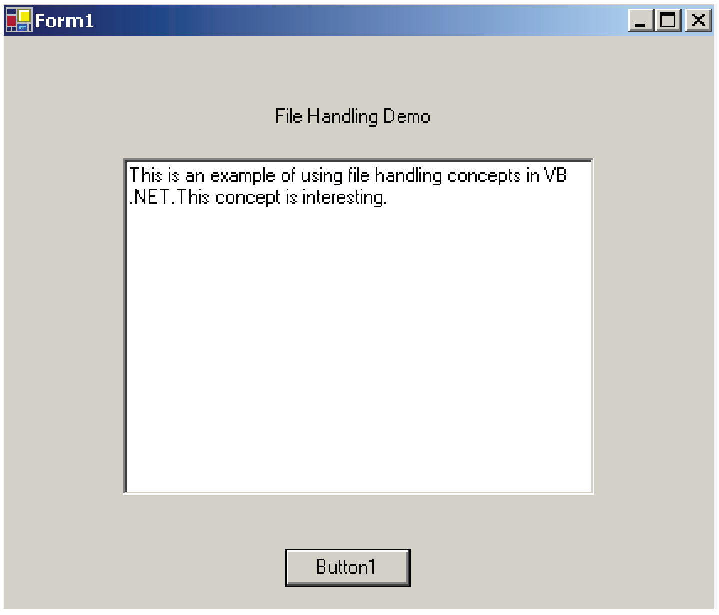
14.2.5 Code to create a file and read from it

Drag a Button and a RichTextBox control onto the form. Paste the following code which is shown below.

```
Imports System.IO
'NameSpace required to be imported to work with files
```

```
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Button1_Click(ByVal....., Byval.....)Handles Button1.Click
Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)
'declaring a FileStream and creating a document file named file with
'access mode of writing
Dim s as new StreamWriter(fs)
'creating a new StreamWriter and passing the filestream object fs as argument
s.WriteLine("This is an example of using file handling concepts in VB .NET.")
s.WriteLine("This concept is interesting.")          'writing text to the newly created file
s.Close()                                       'closing the file
fs=New FileStream("file.doc",FileMode.Open,FileAccess.Read)
'declaring a FileStream to open the file named file.doc with access mode of reading
Dim d as new StreamReader(fs)
'creating a new StreamReader and passing the filestream object fs as argument
d.BaseStream.Seek(0,SeekOrigin.Begin)
'Seek method is used to move the cursor to different positions in a file, in this code, to
'the beginning
while d.peek()>-1
'peek method of StreamReader object tells how much more data is left in the file
RichTextbox1.Text &= d.readLine()
'displaying text from doc file in the RichTextBox
End while
d.close()
End Sub
```

Output of the above code is displayed below.



```
This is an example of using file handling concepts in VB
.NET.This concept is interesting.
```

Self Learning Exercise-1

i. Fill in the Blanks

a) File handling in VB.NET is based on _____________ namespace.

ii. True/False

a) The StreamReader and StreamWriter classes enable us to read or write a random stream of characters to or from a file.

b) The BinaryReader and BinaryWriter classes enable us to read and write binary data in a file.

14.3 Object

All object types inherit, either directly or indirectly, from the CLR type System.Object class. A major facility of the Object class is its ability to enforce a singular rooted inheritance hierarchy on all types in the CLR. Although you may think that this kind of inheritance does not apply to value types, value types can be treated as a subtype of Object through boxing. The libraries make extensive use of the Object type as the parameters to, and the return type of, many functions.

Table 14.1 CLR Built-in Reference Types: Object Types

CIL Name	Library Name	Description	CLS Support
Object	System.Object	Base class for all object types	Y
String	System.String	Unicode string	Y

The Object class provides a number of methods that can be called on all objects:

- Equals returns true if the two objects are equal. Subtypes may override this method to provide either identity or equality comparison. Equals is available as both a virtual method that takes a single parameter consisting of the other object with which this object is being compared and a static method that, naturally, requires two parameters.
- Finalize is invoked by the garbage collector before an object’s memory is reclaimed. Because the garbage collector is not guaranteed to run during a program’s execution, this method may not be invoked. In C#, if a developer defines a destructor, then it is renamed to be the type’s Finalize method.
- GetHashCode returns a hash code for an object. It can be used when inserting objects into containers that require a key to be associated with each such object.
- GetType returns the type object for this object. This method gives access to the metadata for the object. A static method on the Type class can be used for the same purpose; it does not require that an instance of the class be created first.
- MemberwiseClone returns a shallow copy of the object. This method has protected accessibility and, therefore, can be accessed only by subtypes. It cannot be verridden. If a deep copy is required, then the developer should implement the ICloneable interface.
- ReferenceEquals returns true if both object references passed to the method refer to the same object. It also returns true if both references are null.

- ToString returns a string that represents the object. As defined in Object, this method returns the name of the type of the object—that is, its exact type. Subtypes may override this method to have the return string represent the object as the developer sees fit. For example, the String class returns the value of the string and not the name of the String class.

Most of the methods defined on Object are public. MemberwiseClone and Finalize, however, have protected access; that is, only subtypes can access them. The following program output shows the assembly qualified name and the publicly available methods on the Object class

```
System.Object, mscorlib, Version=1.0.2411.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089
Int32 GetHashCode()
Boolean Equals(System.Object)
System.String ToString()
Boolean Equals(System.Object, System.Object)
Boolean ReferenceEquals(System.Object, System.Object)
System.Type GetType()
Void .ctor()
```

The program retrieves the Object class's Type object and then displays its public method's prototypes. The two protected methods are not shown. The preceding output also shows the use of built-in types, such as Boolean, Int32, and String.

Example given below demonstrates how many classes override the ToString method. The default behavior is to print a string representing the type of the object on which it is invoked—that is the action of the Object class. String and Int32 have both overridden this behavior to provide a more intuitive string representation of the object on which it is invoked.

Overriding the ToString method

```
using System;
 namespace Override
{
  class Sample
  {
   static void Print(params Object[] objects)
    {
     foreach(Object o in objects)
      Console.WriteLine(o.ToString());
    }
   static void Main(string[] args)
    {
     Object o = new Object();
    String s = "Mark";
 int i = 42;
 Print(o, s, i);
} } }
```

One interesting feature of this example relates to the use of the params keyword. You can call the Print method with any number of arguments, and these arguments will be passed to the method in an array that holds Object references. Within the Print method, each member of the array will have its virtual ToString method called, so the correct method for each argument will be invoked. You may wonder how the value i of type int is passed given that it is a value type: It is boxed. Listing 2.5 produces the following output:

```
System.Object
Mark
42
```

The constructor for Object should be called whenever an object is created; it must be called if the goal is to produce verifiable code. “verifiable code” means proven type-safe code. As a simplification of the rules, compiler writers must ensure that a call to the constructor for the base class occurs during construction of all derived classes. This call can occur at any time during the construction of derived classes; it need not be the first instruction executed in the constructor of subtype. Developers should be aware that construction of the base class may not have occurred when a user-defined constructor starts running.

Self Learning Exercise-2

i. Fill in the Blanks

b) All object types inherit, either directly or indirectly, grom the CLR type ___________ class.

c) __________ method returns a string that represents the object.

ii. True/False

c) GetHashCode returns a hash code for an object.

d) Equals is available only as static method.

14.4 Error Handling

You will learn how to build blocks of code that handle run time errors, also referred as exceptions which occur as a result of normal operating conditions for example. Errors due to a disk not being in the drive or to an offline printer.

Visual Basic .NET includes the Try—Catch code block, a new syntax for handling error. Now you’ll learn how to trap run time errors using Try—Catch code block, and how to use the Err.Number and Err.Description properties to identify specific runtime errors. You’ll also learn how to use multiple Catch statements to write more flexible error handlers, build nested Try—Catch code blocks, and use the Exit Try statement to exit a Try…Catch code block prematurely.

14.4.1 Processing Errors

A runtime error, or program crash, is an unexpected problem in a Visual Basic program from which the program can’t recover. It’s not that Visual Basic don’t smart enough to handle the glitch; it’s just that Visual Basic hasn’t been told what to do when something goes wrong.

You however do not have to live with the occasional errors that cause your programs to crash.

You can write special Visual Basic runtimes, called structured error handlers, to respond to runtime errors. An error handler handles a runtime error by telling the program how to continue when one of its statements doesn't work. Error handlers are placed in the event procedures, in which there is a potential for trouble, or in generic functions or subprograms that handle errors for you systematically. As their name implies, error handlers handle, or trap, a problem by using the Try…Catch statements and a special error-handling object Err. The Err object has a Number property that identifies the error number and a Description property that allows you to print a description of the error.

14.4.2 When to Use Error Handlers

You can use error handlers in any situation in which an expected or an unexpected action might result in an error that stops program execution. Typically, error handlers are used to process external events that influence a program – for example, events caused by a failed network or Internet connection, a disk not being in the floppy drive, or an offline printer. The Following table lists potential problems that can be addressed by error handlers

Table 14.2: Problems that can be handled by error handlers

Problem	**Description**
Network/Internet Problems	Network servers, modems or resources that fail, or go down, unexpectedly.
Disk drive problems	Unformatted or incorrectly formatted disks, disk that aren't properly inserted, bad disk sectors, disks that are full, problems with CD-ROM drives, and so on.
Path Problems	A path to a necessary file is missing or incorrect.
Printer Problems	Printers that are offline, out of paper, out of memory, or otherwise unavailable.
Software not installed	A file or component that your application relies on is not installed on user's computer, or there's an operating system incompatibility.
Permission Problems	The user doesn't have the appropriate permissions to perform a task.
Overflow Errors	An activity that exceeds the allocated memory space.
Out-of-Memory errors	Application or resource space that's not available in Microsoft Windows.
Clipboard Problems	Problems with data transfer or Windows clipboard.
Logic Errors	Syntax or logic error undetected by the compiler

14.4.3 The Try — Catch Statement

The basic syntax for Try - - - Catch exception handler is simply the following:

The Try Statement identifies the beginning of an error handler:

```
 Try
     'Statements that might produce a runtime error
    Catch
```

```
  'Statements to run if runtime errors occur
 Finally
  'Optional statements to run whether an error occur or not
 End Try
```

The statement between the Try – Catch keywords is called protected code because any runtime errors resulting from these statements won't cause the program to crash.

Example:

```
   Try
   prcTextFile.StartInfo.FileName = ("C:\ProgramFiles\ErrorHandler\VbNetError.txt ")
     prcTextFile.Start ()
  Catch
    MessageBox.Show ("Unable to locate the desired file")
  Finally
    MessageBox.Show ("Error Handler Complete")
  End Try
```

In a real program you'll probably want to use the Finally code block to update important variables or properties, display data, or perform other cleanup operations.

14.4.4 More Complex Try — Catch Error Handlers

As your program become more sophisticated, you might find it useful to write more complex Try —_ Catch error handlers that manage a variety of runtime errors and unusual error handling situations. Try — Catch provides for this complexity by:

(1). Permitting multiple lines of code in each Try, Catch or Finally code block.

(2). Offering the Catch When syntax, which tests specific error conditions.

(3). Allowing nested Try — Catch code block, which can be used to build sophisticated and robust error handlers.

In addition a special error handling object named Err allows you to identify and process runtime errors and conditions in your program.

14.4.5 The *Err* Object

Err is a special Visual Basic object that's assigned detailed error handling information each time a runtime error occurs. The most useful Err properties for identifying runtime errors are Err.Number and Err.Description.

Err.Number: contains the number of the most recent runtime error.

Err.Description: a short error message that matches the runtime error numbers.

By using the Err.Number and Err.Description properties together in an error handler, you can recognize specific errors and respond to them and you can give the user helpful information about how he or she should respond.

The Following table lists a few of the runtime errors that Visual Basic application can encounter:

Table 14.3: Runtime Errors

Error Number	Default Error Message
5	Procedure call or argument is not valid
6	Overflow
7	Out of Memory
11	Division by Zero
51	Internal Error
52	Bad file name or number
53	File not found
55	File already open
76	Path not found
482	Printer error

For more information on a particular error, search for Visual Studio online Help. Unused error numbers in the range of 1-1000 are reserved for future use by Visual Basic.NET.

14.4.6 Test for Multiple Runtime Errors:

```
Try
      prcTextFile.StartInfo.FileName = (“C:\Program Files\ErrorHandler\VbNetError.txt “)
       prcTextFile.Start ()
    Catch When Err. Number = 53 ‘If File Not Found Error
      MessageBox.Show (“Unable to locate the desired file”)
    Catch When Err. Number = 7 ‘If Out of Memory Error
      MessageBox.Show (“Are you sure about the file Path”)
     Catch
      MessageBox.Show (“Problem Loading File”)
     Finally
      MessageBox.Show (“Error Handler Complete”)
     End Try
```

14.4.7 Exit Try

As with any block structure, it is very nice to be able to jump out of the structure when needed. This is the purpose of the Exit Try statement.

If we have a Finally block that block’s code must be run before the Try block is exited.

```
      Try
        ‘Any Protected Code
      prcTextFile.StartInfo.FileName = (“C:\Program Files\ErrorHandler\VbNetError.txt “)
       prcTextFile.Start ()
       If blnFlg = True Then Exit Try
     Catch
       ‘Error Handling Logic/Code
```

```
    MessageBox.Show ("Unable to locate the desired file")
  Finally
    'Execution Resumes Here
    MessageBox.Show ("Error Handler Complete")
  End Try
```

Based on the flow of your code, you can jump out of the protected region at any point with the Exit Try statement.

14.4.8 Error Handlers and Defensive Programming Techniques

Consider the following piece of code:

```
If File exists Then
    prcTextFile.StartInfo.FileName = ("C:\Program Files\ErrorHandler\VbNetError.txt ")
     prcTextFile.Start ()
  Else
    MessageBox.Show ("File Not Found")
  End If
```

This If—Then statement isn't an actual error handler because it doesn't prevent a runtime error from halting a program. Instead, it's a validation technique that some programmers call Defensive Programming.

In this particular case, testing to see whether a file exists with a .NET framework method is actually faster than waiting for Visual Basic to issue an exception and recover from an runtime error using an error handler. When should you use defensive programming techniques and when should you use error handlers? The answer depends on how often you think a problem will occur with the statements you plan to use. If there's a real likelihood that a piece of code will produce a runtime error more than 25 percent time, defensive programming logic is usually the most efficient way to manage potential problems.

Self Learning Exercise-3

i. Fill in the Blanks

d) _________is a special object that is assigned detailed error handling information.

ii. True/False

e) Syntax for the Try—Catch is as follows

```
Try
'statements
Finally
'statements
Catch
'statements
End Try
```

f) The most useful Err properties for identifying runtime errors are Err.Number and Err.Description.

14.5 Multithreading

Multithreading gives programs the ability to do several things at a time. Each stream of execution is called a thread. Multithreading is used to divide lengthy tasks into different segments that would otherwise abort programs. Threads are mainly used to utilize the processor to a maximum extent by avoiding its idle time. Threading lets a program seem as if it is executing several tasks at once. What actually happens is, the time gets divided by the computer into parts and when a new thread starts, that thread gets a portion of the divided time. Threads in VB .NET are based on the namespace System.Threading.

14.5.1 Creating Threads

To create threads lets work with an example. The following example opens a new windows application and name it as Thread and add a class named count1 using the Projects->Add Class item. This class will count from 1 to a specified value in a data member named CountTo when you call the Count method. After the count has reached the value in CountTo, a FinishedCounting event will occur. The code for the Count class looks like this:

```
Public Class Count1
Public CountTo as Integer
Public event FinishedCounting(By Val NumberOfMatches as Integer)
Sub Count()
Dim ind,tot as Integer
tot=0
For ind=1 to CountTo
tot+=1
Next ind
RaiseEvent FinishedCounting(tot)
'makes the FinishedCounting event to occur
End Sub
End Class
```

Let us use this class with a new thread. Get back to the main form and create an object of this class, counter1, and a new thread, Thread1. The code looks like this:

```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim counter1 as new Count1()
Dim Thread1 as New System.Threading.Thread(Address of counter.Count)
```

Drag a Button and two TextBoxes (TextBox1, TextBox2) onto the form. Enter a number in TextBox1. The reason for entering a number in textbox is to allow the code to read the value specified in TextBox1 and display that value in TextBox2, with threading. The code for that looks like this:

```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim counter1 as new Count1()
Dim Thread1 as New System.Threading.Thread(Address of counter.Count)
Private Sub Button1_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button1.Click
TextBox2.Text=" "
counter1.CountTo=TextBox1.Text
AddHandler counter1.FinishedCounting,AddressOfFinishedCountingEventHandler
'adding handler to handle FinishedCounting Event
Thread1.Start()
'starting the thread
End Sub
Sub FinishedCountingEventHandler(ByVal Count as Integer)
'FinishedCountingEventHandler
TextBox2.Text=Count
End Sub
```

The result of the above code displays the value entered in TextBox1, in TextBox2 with the difference being the Thread counting the value from 1 to the value entered in TextBox1.

14.5.2 Suspending a Thread

Threads can be suspended. Suspending a thread stops it temporarily. Working with the example in the previous section, add a new button Button2 to the main form. When this button is clicked the thread is suspended. The code for that looks like this:

```
Private Sub Button2_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button2.Click
Thread1.Suspend()
End Sub
```

14.5.3 Resuming a Thread

Threads can be resumed after they are suspended. With the example above, add a new button Button3 to the main form. When this button is clicked the thread is resumed from suspension. The code for that looks like this:

```
Private Sub Button3_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button3.Click
Thread1.Resume()
End Sub
```

14.5.4 Making a Thread Sleep

Threads can be made to sleep which means that they can be suspended over a specific period of time. Sleeping a thread is achieved by passing the time (in milliseconds,1/1000 of a second) to the thread's sleep method. With the example above, add a new button Button4 to the main form. When this button is clicked the thread is stopped. The code for that looks like this:

```
Private Sub Button4_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button4.Click
```

```
Thread1.Sleep(100/1000)
End Sub
```

14.5.5 Stopping a Thread

Threads can be stopped with it's abort method. With the example above, add a new button Button5 to the main form. When this button is clicked the thread is stopped. The code for that looks like this:

```
Private Sub Button5_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button5.Click
Thread1.Abort()
End Sub
```

14.5.6 Thread Priorities

Threads can also be assigned priority for execution. Thread priority can be set by the thread's Priority property and assigning a value from predefined Thread Priority enumeration.

Values for Thread Priority:

Above Normal -> Gives thread higher priority

Below Normal ->Gives thread lower priority

Normal -> Gives thread normal priority

Lowest -> Gives thread lowest priority

Highest -> Gives thread highest priority

Working with the above example, add a new button Button6 to the main form. When this button is clicked the thread is assigned Highest priority .The code for that looks like this:

```
Private Sub Button6_Click(ByVal sender as System.Object, ByVal e as System.EventArgs)_
Handles Button6.Click
Thread1.Priority=System.Threading.ThreadPriority.Highest
'setting Highest priority for the thread
End Sub
```

Self Learning Exercise-4

i. Fill in the Blanks

e) _______ feature allows the program to do several things at a time.

f) Sleeping of a thread is achieved by passing the time in _______________ to the thread's sleep method.

ii. True/False

f) Threads can be stopped with its Stop() method.

g) Threads can't assign the priority.

14.6 Data Access

14.6.1 Data Grid

One of the most common tasks in classic ASP was retrieving tabular information from a database and displaying it in an HTML table. With classic ASP this would require many lines of intermixed HTML and code; the following pseudocode shows what the code would commonly look like:

```
Create Database Connection
Populate a recordset based on some SQL query
Output the HTML table header (<table ...>)
Loop through the recordset
  Emit the HTML for a table row
  ...
Emit the HTML table footer (</table>)
```

One of the advantages of ASP.NET is that it contains a number of Web controls. These Web controls, which emit HTML, provide a programmatic interface, allowing developers to separate code and content and treat these HTML emitting entities as objects in their code. That is, if we wanted to display some HTML content using ASP.NET we'd do the following:

```
<script language="vb" runat="server">
  sub Page_Load(sender as Object, e as EventArgs)
    lblMessage.Text = "Hello, World!"
  end sub
</script>
<asp:label runat="server" id="lblMessage" />
```

Here the label Web control lblMessage is placed in the HTML using HTML-like tags with the runat="server" attribute specified. Then, in the Page_Load event handler (which is run every time the page is loaded) the Text property of the lblMessage Web control is set to "Hello, World!" The use of Web controls here separates the code from the content; in classic ASP one would need to place a <%="Hello, World!"%> in the proper place within the HTML content to achieve the same effect.

There are ASP.NET Web controls that are much more useful and powerful than the simple label control. The DataGrid Web control is one such powerful control. The DataGrid emits the needed HTML for data-bound HTML tables. As we'll soon see, binding data to a DataGrid is very easy; furthermore, with only a few slight property changes you can customize the look and feel the DataGrid's output, rendering very professional looking HTML tables.

14.6.1.1 DataGrid Basics

To place a DataGrid on an ASP.NET Web page you need to add the following code:

```
<asp:datagrid runat="server" id="ID_of_DataGrid" />
```

Here the id you choose will be the name of the DataGrid you'll use when referring to it in your server-side code. The syntax above gets us started using a DataGrid by placing it in the HTML content, but in order to have the DataGrid display anything useful we need to bind the DataGrid to some collection of information. This collection of information can be any object that supports the IEnumerable interface. This includes things like Arrays, collection classes (ArrayList, Hashtable, etc.), DataSets, DataReaders, and a number of other objects. Since we'd like to focus on displaying database information, for this article we'll focus on binding DataGrids to DataReaders, which are synonymous to forward-only, firehose cursors Recordsets in classic ADO/ASP. (For more information on reading database results into DataReaders using ADO.NET be sure to read: Efficiently Iterating Through Results from a Database Query using ADO.NET.)

So, how is data bound to the DataGrid? The first thing we need to do is to take a DataReader containing some database data. For this example we hitting the ASPFAQs.com database and bring back the ten most popular FAQs. Once we have this data in a DataReader, to bind it to our DataGrid we just need to write two lines of code. The first line sets the DataGrid's DataSource property to the DataReader; the second line calls the DataGrid's DataBind method.

```
<% @Import Namespace=”System.Data” %>
<% @Import Namespace=”System.Data.SqlClient” %>
<script language=”vb” runat=”server”>
 Sub Page_Load(sender as Object, e as EventArgs)
  BindData()
 End Sub
 Sub BindData()
  ‘1. Create a connection
  Dim myConnection as New SqlConnection(
     ConfigurationSettings.AppSettings(“connectionString”))

  ‘2. Create the command object, passing in the SQL string
  Const strSQL as String = “sp_Popularity”
  Dim myCommand as New SqlCommand(strSQL, myConnection)
  ‘Set the datagrid’s datasource to the datareader and databind
  myConnection.Open()
  dgPopularFAQs.DataSource = myCommand.ExecuteReader(
                    CommandBehavior.CloseConnection)
  dgPopularFAQs.DataBind()
 End Sub
</script>
<asp:datagrid id=”dgPopularFAQs” runat=”server” />
```

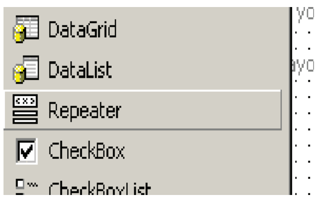
Note that the amount of code we have to write to utilize databinding is not much at all. We essentially create a connection, specify a SQL command (in this case a stored procedure, sp_Popularity), open the database connection, set the DataGrid's DataSource property to the resulting DataReader, and finally call the DataGrid's DataBind method. This approach completely isolates the code from the content; there's no mixing of HTML table and DataReader output syntax, as we would have had with classic ASP

The DataGrid control is a versatile control and allows for a high degree of customization. However, if you want to simply display items from a data source as an HTML table, there is a simpler control—the Repeater control.

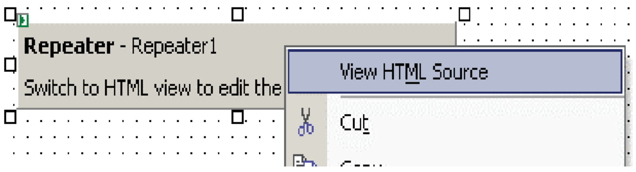
14.6.2 Repeater Control

The Repeater control performs a very common function that most Web developers have encountered in their projects. Very often you need to display records from a database, and most likely you would use a FOR loop to write the HTML code so that the records can be displayed within a table. Using the Repeater control, this process can very easily be automated.

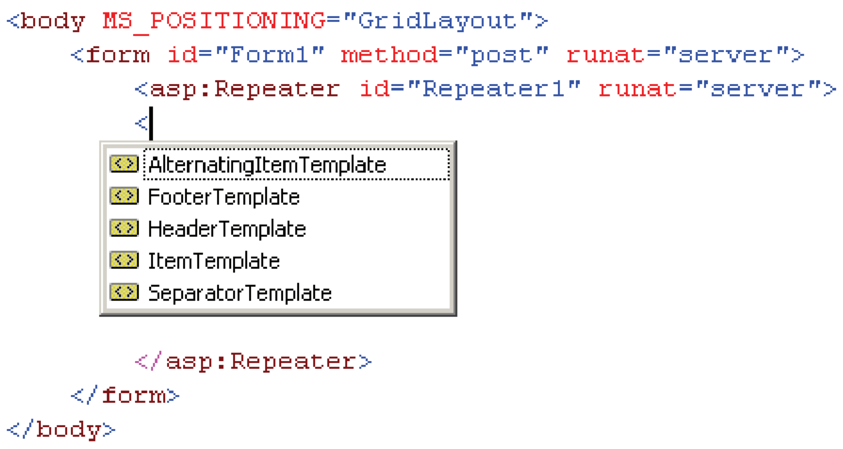
The repeater control can be found within the Toolbox in Visual Studio .NET:



Simply drag and drop the control onto the Web Form. To customize the Repeater control, switch to HTML view, as shown below.

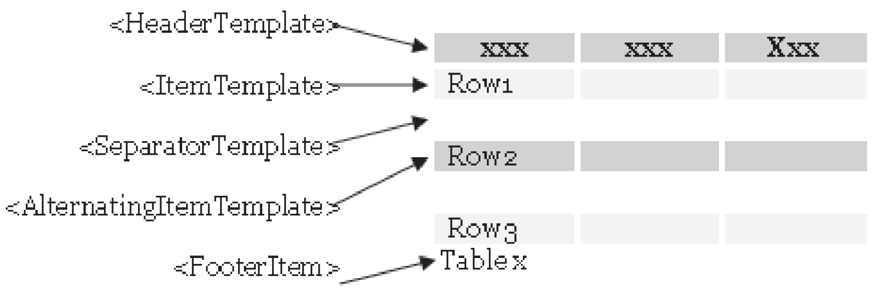


Note that the Repeater control contains five templates to customize the behavior of the control:



```
<body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
        <asp:Repeater id="Repeater1" runat="server">
        <|
        </asp:Repeater>
    </form>
</body>
```

A good way to understand the use of these five templates is to refer to the following figure.



When the Repeater control is bound to a data source, the rows within the data source will be displayed by the <ItemTemplate> and <AlternatingItemTemplate> (if present) elements.

Let’s now add the <HeaderTemplate>, <ItemTemplate>, <SeparatorTemplate>, and <AlternatingItemTemplates> elements to our Repeater control:

```
<asp:Repeater id=”Repeater1" runat=”server”>
  <HeaderTemplate>
     <table border=”1">
       <tr bgcolor=”#ffcc99">
          <th>ID</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Address</th>
       </tr>
  </HeaderTemplate>
  <ItemTemplate>
    <tr bgcolor=”#ffcccc”>
     <td><%# DataBinder.Eval(Container.DataItem, “au_id”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “au_fname”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “au_lname”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “address”) %></td>
     </tr>
  </ItemTemplate>
  <AlternatingItemTemplate>
    <tr bgcolor=”#ccff99">
     <td><%# DataBinder.Eval(Container.DataItem, “au_id”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “au_fname”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “au_lname”) %></td>
     <td><%# DataBinder.Eval(Container.DataItem, “address”) %></td>
     </tr>
  </AlternatingItemTemplate>
  <FooterTemplate>
            </table>
  </FooterTemplate>
</asp:Repeater>
```

We then bind the Repeater control to a DataSet. To populate each row of the table with a record from the DataSet used:

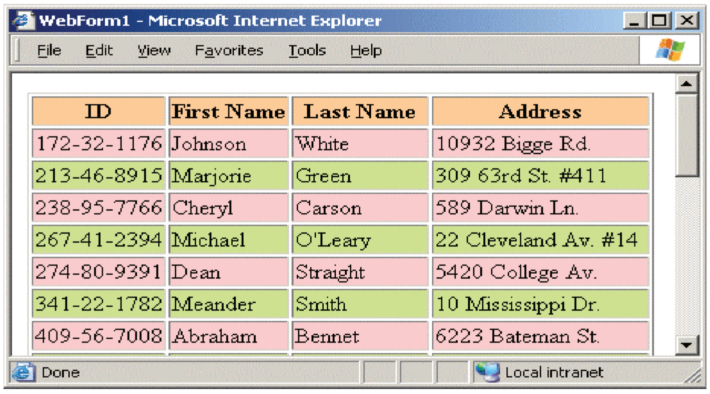
```
<%# DataBinder.Eval(Container.DataItem, “au_id”) %>
```

In this case, “au_id” refers to the name of a column in the DataSet.

Finally, add a SqlDataAdapter control to your project and connect it to the Authors table within the Pubs database. In the Page_Load event, add in the following codes:

```
  Private Sub Page_Load(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MyBase.Load
      If Not IsPostBack Then
          Dim ds As New DataSet
        SqlDataAdapter1.SelectCommand.CommandText = _
         "SELECT * FROM Authors"
        SqlDataAdapter1.Fill(ds, "employee_record")
         Repeater1.DataSource = ds
         Repeater1.DataBind()
      End If
  End Sub
```

Run the application. You should now see the following:


WebForm1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

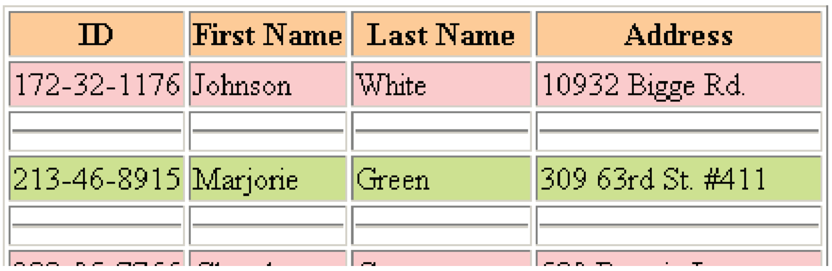
ID	First Name	Last Name	Address
172-32-1176	Johnson	White	10932 Bigge Rd.
213-46-8915	Marjorie	Green	309 63rd St. #411
238-95-7766	Cheryl	Carson	589 Darwin Ln.
267-41-2394	Michael	O'Leary	22 Cleveland Av. #14
274-80-9391	Dean	Straight	5420 College Av.
341-22-1782	Meander	Smith	10 Mississippi Dr.
409-56-7008	Abraham	Bennet	6223 Bateman St.

The <separatortemplate> element allows you to inject a separator between rows of records. For example, you might wish to insert a horizontal rule between records, as the following shows:

```
 ...
 </ItemTemplate>
 <SeparatorTemplate>
    <tr>
      <td><hr></td>
      <td><hr></td>
      <td><hr></td>
      <td><hr></td>
    </tr>
 </SeparatorTemplate>
 <AlternatingItemTemplate>
```

...

Now, all the records are separated by a horizontal rule (<hr>):



ID	First Name	Last Name	Address
172-32-1176	Johnson	White	10932 Bigge Rd.
213-46-8915	Marjorie	Green	309 63rd St. #411

Adding Button Controls to a Repeater

You can also add Button controls to a Repeater. For example, you might use a Button control to represent an Author ID, and when the button is clicked, display more information about the author.

Let's modify our HTML source to include a Button control. Change the line (there are a total of two such lines in our HTML source):

```
<td><%# DataBinder.Eval(Container.DataItem, "au_id") %></td>
```

to:

```
<td><asp:Button CommandName="ID"
    Text=<%# DataBinder.Eval(Container.DataItem, "au_id") %>
    runat=server/></td>
```

The above line will add a Button control within the Repeater control. The CommandName property is used to uniquely identify the button; if there is more than one Button control, each control must have a different CommandName.

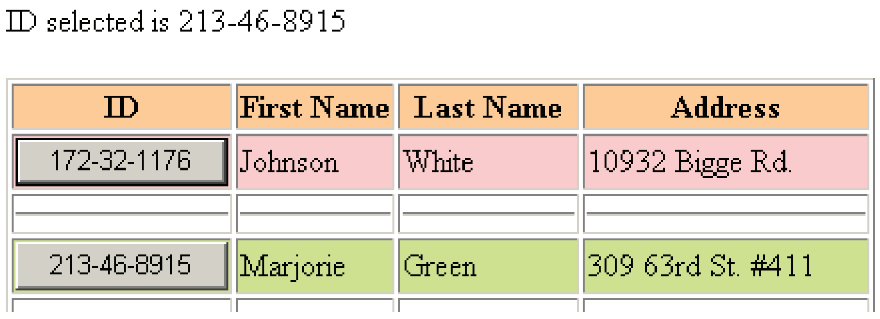
Next, we need to modify the <asp:Repeater> element so that it knows which event to fire when the Button control is clicked. We need to specify the OnItemCommand property, indicating the event to service when the button is clicked:

```
<asp:Repeater id="Repeater1"
 OnItemCommand="Button_ItemCommand" runat="server">
```

Lastly, code the routine for the Button_ItemCommand event:

```
  Sub Button_ItemCommand(ByVal Sender As Object, _
                  ByVal e As RepeaterCommandEventArgs)
     If e.CommandName = "ID" Then
        Response.Write("ID selected is " & _
          CType(e.CommandSource, Button).Text())
     End If
  End Sub
```

Run the application and click on one of the buttons:


ID selected is 213-46-8915

ID	First Name	Last Name	Address
172-32-1176	Johnson	White	10932 Bigge Rd.
213-46-8915	Marjorie	Green	309 63rd St. #411

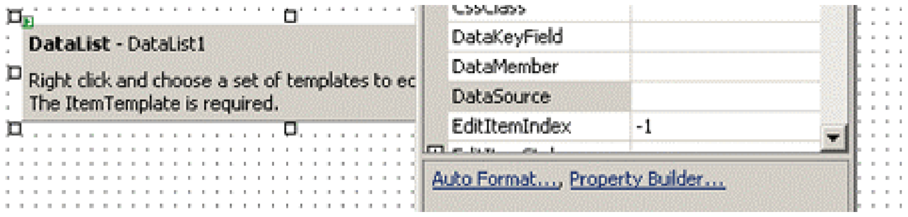
Note that in the Form_Load event, you need to ensure that subsequent postbacks do not result in another databinding process. If you do not test for postbacks using the IsPostBack property, the Button_ItemCommand event will not be serviced.

14.6.3 Data List

The DataList control is somewhat a combination of the DataGrid and Repeater controls. It works like the Repeater control, allowing you to create templates so that it can be bound to a data source. It also allows you to edit records, much like the DataGrid control.

To use the DataList control, drag and drop the DataList control from the Toolbox in Visual Studio .NET.

You can specify the formatting of the DataList control by right-clicking on the control and selecting Properties. In the Properties window, click on the Auto Format... link at the bottom:


DataList - DataList1

Right click and choose a set of templates to ec
The ItemTemplate is required.

DataKeyField	
DataMember	
DataSource	
EditItemIndex	-1

Choose the color scheme that you like. When you are done, switch to HTML view mode. You should see something like this:

```
<asp:DataList id=”DataList1"
  style=”Z-INDEX: 101;
    LEFT: 16px;
    POSITION: absolute;
    TOP: 16px”
  runat=”server”
  BorderColor=”#DEBA84" BorderStyle=”None”
  CellSpacing=”2" BackColor=”#DEBA84"
  CellPadding=”3" GridLines=”Both” BorderWidth=”1px”>
  <SelectedItemStyle Font-Bold=”True” ForeColor=”White”
```

```
    BackColor="#738A9C">
  </SelectedItemStyle>
  <ItemStyle ForeColor="#8C4510" BackColor="#FFF7E7">
  </ItemStyle>
  <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
  </FooterStyle>
  <HeaderStyle Font-Bold="True" ForeColor="White"
    BackColor="#A55129">
  </HeaderStyle>
</asp:DataList>
```

The DataList control contains seven templates and seven styles

The DataList control works quite similar the Repeater control (see my previous article on the Repeater control for a description of the various templates). However, it also supports the Edit, Update, and Cancel features found in the DataGrid control.

Let's modify the HTML code for our control, and each addition will be explained along the way.

```
<asp:DataList id="DataList1"
   style="Z-INDEX: 101;
          LEFT: 16px;
          POSITION: absolute;
          TOP: 16px"
         runat="server"
   CellPadding="3" BackColor="#DEBA84" BorderWidth="1px"
   CellSpacing="2" BorderStyle="None" BorderColor="#DEBA84"
   GridLines="Both"
```

Add the following attributes to the <asp:DataList> element:

```
   OnEditCommand="Edit_Command"
   OnCancelCommand="Cancel_Command"
   OnUpdateCommand="Update_Command"
   OnDeleteCommand="Delete_Command"
   DataKeyField="title_id">
```

Later, we will add a few LinkButton controls to our DataList control to perform the functions of Edit, Update, Delete, and Cancel. The above attributes specify the methods to invoke when such LinkButton controls are clicked. The key of this control to be the field title_id has also been defined.

Next we add a <HeaderTemplate> element. The text within this element will be displayed as the title of the table.

```
  <HeaderTemplate>
    Titles
```

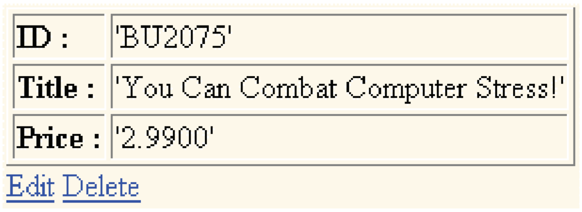
```
</HeaderTemplate>
```

The <ItemTemplate> element allows you to display records from a database by binding them to the control. Here, three fields (title_id, title, and price) from the Titles table (we will see this shortly) are. displayed.

```
<ItemTemplate>
   <table border=1>
   <tr><td><b>ID : </b></td>
     <td>'<%# DataBinder.Eval(Container.DataItem, "title_id") %>'
       </td></tr>
  <tr><td><b>Title : </b></td>
     <td>'<%# DataBinder.Eval(Container.DataItem, "title") %>'
       </td></tr>
   <tr><td><b>Price : </b></td>
     <td>'<%# DataBinder.Eval(Container.DataItem, "price") %>'
   </td></tr>
   </table>
```

We have also added two LinkButton controls so that the user can edit or delete the record:

```
          <asp:LinkButton Text="Edit" CommandName="Edit"
                Runat="server" ID="edit" />
   <asp:LinkButton Text="Delete" CommandName="Delete"
                Runat="server" ID="delete" />
</ItemTemplate>
```

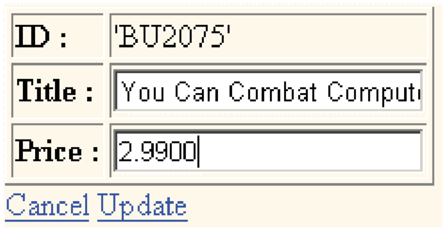


ID :	'BU2075'
Title :	'You Can Combat Computer Stress!'
Price :	'2.9900'

Next, add in the <EditItemTemplate> element so that textboxes can be displayed when the Edit button is clicked. Also add two LinkButton controls to display the Update and Cancel buttons:

```
<EditItemTemplate>
   <table border=1>
   <tr><td><b>ID : </b></td>
```

```
      <td>'<%# DataBinder.Eval(Container.DataItem, "title_id") %>'
       </td></tr>
    <tr><td><b>Title : </b></td>
       <td><asp:TextBox
         Text='<%# DataBinder.Eval(Container.DataItem, "title") %>'
         runat=server ID="title"/>
       </td></tr>
    <tr><td><b>Price : </b></td>
       <td><asp:TextBox
         Text='<%# DataBinder.Eval(Container.DataItem, "price") %>'
         runat=server ID="price"/></td></tr>
    </table>
    <asp:LinkButton Text="Cancel" CommandName="Cancel"
                Runat="server" ID="cancel" />
    <asp:LinkButton Text="Update" CommandName="Update"
                Runat="server" ID="update" />
  </EditItemTemplate>
```



ID :	'BU2075'
Title :	You Can Combat Comput
Price :	2.9900

That's it! We now write the code to perform the data binding as well as the editing of the records. First, import the relevant namespace for data access:

Imports System.Data.SqlClient

When the page is loaded for the first time, load the records from the database:

```
  Private Sub Page_Load(ByVal sender As System.Object, _
                 ByVal e As System.EventArgs) _
                                        Handles MyBase.Load
     'Put user code to initialize the page here
     If Not IsPostBack Then
        LoadData()
     End If
  End Sub
```

The LoadData() method binds the records from the Titles table (from the Pubs database) to the DataList control. Be sure to add a SqlDataAdapter control to your project.

```
  Public Sub LoadData()
     Dim ds As New DataSet
    SqlDataAdapter1.SelectCommand.CommandText = _
    "SELECT * FROM Titles"
    SqlDataAdapter1.Fill(ds, "titles_record")
    DataList1.DataSource = ds
    DataList1.DataBind()
  End Sub
```

When the Edit link is clicked, you need to set the index for the record to be edited:

```
  Sub Edit_Command(ByVal sender As Object, _
                    ByVal e As DataListCommandEventArgs)
    ' to rebind the DataList to the data source to
                   ' refresh the control.
    DataList1.EditItemIndex = e.Item.ItemIndex
     LoadData()
  End Sub
```

When the Cancel link is clicked, reset the edit index to -1:

```
  Sub Cancel_Command(ByVal sender As Object, _
    ByVal e As DataListCommandEventArgs)
    DataList1.EditItemIndex = -1
     LoadData()
  End Sub
```

When the Update link is clicked, you need to update the relevant record. Here we extract the key of the control (which is the title_id) and the values in the edit textbox:

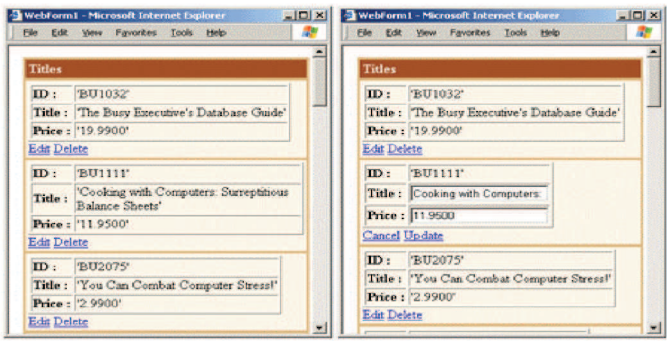
```
  Sub Update_Command(ByVal sender As Object, _
                ByVal e As DataListCommandEventArgs)
    Dim tbox As TextBox
    Dim title_id, title As String
    Dim price As Single
    '—retrieves the key for the row—
    title_id = DataList1.DataKeys(e.Item.ItemIndex)
    '—find the textbox control containing the title
    tbox = CType(e.Item.FindControl("title"), TextBox)
    title = tbox.Text
    '—find the textbox control containing the price
```

```
        tbox = CType(e.Item.FindControl("price"), TextBox)
         price = tbox.Text
         '—updates the database—
        Dim sql As String = "UPDATE titles SET title='" & _
                          title & "' , price=" & price & _
                        " WHERE title_id='" & title_id & "'"
        Dim conn As New SqlConnection("server=localhost; " & _
                       "user id =sa; password=;database=pubs")
        Dim comm As New SqlCommand(sql, conn)
         conn.Open()
        comm.ExecuteNonQuery()
         conn.Close()
        DataList1.EditItemIndex = -1
         LoadData()
     End Sub
```

Finally, you need to add the code for deleting a record.

```
   Sub Delete_Command(ByVal sender As Object, _
                  ByVal e As DataListCommandEventArgs)
        '—retrieves the key for the row—
        Response.Write(DataList1.DataKeys(e.Item.ItemIndex))
         '—codes to delete row here——
          '
          '_________________________________
     End Sub
```

Press F5 and run the application. You should see something like this:


WebForm1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
WebForm1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Titles

ID : 'BU1032'
Title : 'The Busy Executive's Database Guide'
Price : '19.9900'
Edit Delete

ID : 'BU1111'
Title : 'Cooking with Computers: Surreptitious Balance Sheets'
Price : '11.9500'
Edit Delete

ID : 'BU2075'
Title : 'You Can Combat Computer Stress!'
Price : '2.9900'
Edit Delete

Titles

ID : 'BU1032'
Title : 'The Busy Executive's Database Guide'
Price : '19.9900'
Edit Delete

ID : 'BU1111'
Title : Cooking with Computers:
Price : 11.9500
Cancel Update

ID : 'BU2075'
Title : 'You Can Combat Computer Stress!'
Price : '2.9900'
Edit Delete

Self Learning Exercise-5

i. Fill in the Blanks

g) The DataList control is somewhat a combination of the DataGrid and ________ controls.

True/False

i) The DataGrid control allows for a high degree of customization.

j) The DataList control contains seven templates and six styles.

14.7 Summary

In this unit we have discussed how to create a file and how to write the data in file after that you can also read the data from a file. You can also introduce the threads in your programs by multithreading. You can also trap a run time error and can handle it. You can display the data from database in your web page using Data Grid, Repeater, and Data List control.

14.8 Glossary

BinaryReader class	Enables us to read binary data, raw 0's and 1's, the form in which data is stored on the computer.
BinaryWriter class	Enables us to write binary data.
DataGrid	Can hold text data, but not linked or embedded objects.
DataList	Control is somewhat a combination of the DataGrid and Repeater controls.
DataRepeater	Can hold other controls and can embed objects.
Err	Err is a special Visual Basic object that's assigned detailed error handling information each time a runtime error occurs.
Error	A Run time Exception.
FileStream Class	Provides access to standard input and output files.
Multithreading	Gives programs the ability to do several things at a time.
Object class	Provides a number of methods that can be called on all objects.
StreamReader class	Enables us to read a sequential stream of characters from a file.
StreamWriter class	Enables us to write a sequential stream of characters to a file.
Threads	Are mainly used to utilize the processor to a maximum extent by avoiding it's idle time.

14.9 Further Readings and References

1. Mridula Parihar, Essam Ahmed, Jim Chandler, Bill Hatfield ,"ASP.NET Bible", Unique Color carton, New Delhi
2. Mathew Macdonald, "ASP.NET: The Complete Reference", Tata Mcgraw Hill, New Delhi
3. John Alexander, Billy Hollis, "Developing Web Applications with Visual Basic .NET and ASP.NET", John Wiley New Delhi
4. http://www.informit.com
5. www.ondotnet.com

14.10 Answers to Self Learning Exercises

i. Fill In the Blanks	ii. True/False
a) System.IO	a) False
b) System.Object	b) True
c) ToString	c) True
d) Err	d) False
e) Multithreading	e) False
f) Milliseconds	f) True
g) Repeater	g) False
	h) False
	i) True
	j) False

14.11 Unit End Questions

1. What are error handling routines explain where and when should use the error handlers.
2. What do you understand by multithreading? Explain with an example.
3. What is the advantage of DataList control over DataGrid and Repeater control?
4. List out any five runtime errors that can be handled .NET?
5. Write a program that will write the data “hello welcome to world of computers” in a file named as file1.doc and retrieve the contents of the file.