



ALAGAPPA UNIVERSITY

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

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

KARAIKUDI – 630 003



Directorate of Distance Education

M.Sc. [Computer Science]

II - Semester

341 23

.NET PROGRAMMING

Authors

Narayan Kasat , Project Manager & Training Facilitator, Radix Info Solutions, Delhi

Units (1, 2.3, 11.6, 12-14)

Rohit Khurana, CEO, ITL Education Solutions Ltd.

Units (3.0-3.6, 3.8-3.12, 4.0-4.3, 5, 7.0-7.2.11, 7.4-7.8)

Dr. Kuldeep Singh Kaswan, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Uttar Pradesh

Dr. Pradeep Tomar, Assistant Professor, Department of Computer Science and Engineering, School of Information and Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh

Units (3.7, 7.2.12-7.2.15, 8, 11.0-11.5, 11.7-11.12)

Dr. Richa Bhargava, Adjunct Faculty, ICFAI Business School (IBS), Gurgaon

Units (6, 10)

Vikas® Publishing House: Units (2.0-2.2, 2.4-2.8, 4.4-4.9, 7.3, 9)

"The copyright shall be vested with Alagappa University"

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

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



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

VIKAS® PUBLISHING HOUSE PVT. LTD.

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

Phone: 0120-4078900 • Fax: 0120-4078999

Regd. Office: 7361, Ravindra Mansion, Ram Nagar, New Delhi 110 055

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

Work Order No. AU/DDE/DE1-291/Preparation and Printing of Course Materials/2018 Dated 19.11.2018 Copies - 500

SYLLABI-BOOK MAPPING TABLE

.NET Programming

Syllabi	Mapping in Book
BLOCK 1: .NET FRAMEWORKS 1. Introduction: CLR, Namespace, Assemblies, Class Library 2. Basic Terminology: .NET Component, .NET Garbage Collection 3. OOPs Concept: Class, Objects, Structures, Modules, Abstraction, Encapsulation, Inheritance, Polymorphism, Overloading, Overriding, Shadowing	Unit 1: Introduction to .NET Framework (Pages 1-18); Unit 2: Basic Terminologies (Pages 19-34); Unit 3: OOP's Concept (Pages 35-72)
BLOCK 2 : VISUAL BASIC.NET 4. Introduction: Data Types, Operators, Arrays, Dynamic Arrays, String Handling 5. Control Statements: Conditional and Looping Statements, Sub Procedures and Functions 6. Windows Forms: MDI Form, Events, msgbox, inputbox, Dialogboxes, Passing Forms, RichTextBoxes, Labels, Link Labels	Unit 4: Introduction to VB.NET (Pages 73-100); Unit 5: Control Statements (Pages 101-125); Unit 6: Windows Forms (Pages 126-156)
BLOCK 3 : WINDOWS CONTROLS 7. Introduction: Buttons, Checkbox, Radio Buttons, Panel, List Boxes, Combo Boxes, Scrollbars, Splitters, Track Bars, Pickers, Notify Icons, Timers, Menus 8. Tree and List View: Toolbars, Status Bars, Progress Bars, Tab Controls 9. Debugging and Error Handling: Types of Errors, Exceptions and Structured Exception Handling	Unit 7: Introduction to Window Controls (Pages 157-181); Unit 8: Tree and ListView (Pages 182-203) Unit 9: Debugging and Error handling (Pages 204-222)
BLOCK 4 : ASP.NET 10. Introduction: File Types, Importing Namespaces, Usage of Global.asax File, The Page Class, HttpRequest, HttpResponse, Server Utility 11. Basic Web Controls: List Controls, Validation and Rich Controls, Data Controls, Custom Controls 12. Overview of AJAX Controls	Unit 10: Introduction to ASP.NET (Pages 223-245); Unit 11: Basic Web Controls (Pages 246-301); Unit 12: Overview of AJAX (Pages 302-325)
BLOCK 5: ADO.NET 13. Introduction: Database Access in the Internet World, Characteristics, Data Objects, Data Namespace 14. SQL Basics: Data Binding Controls, Data Set, Data Table, Data Row, Data Column, Data List, Data Grid	Unit 13: Introduction to Database Access (Pages 326-344); Unit 14: SQL Basics (Pages 345-366)

CONTENTS

INTRODUCTION

BLOCK I: .NET FRAMEWORKS

UNIT 1 INTRODUCTION TO .NET FRAMEWORK 1-18

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Evolution of Web Development
 - 1.2.1 ASP.NET: An Overview
 - 1.2.2 Platform Requirements
- 1.3 The Microsoft .NET Framework
- 1.4 .NET Framework Class Library
 - 1.4.1 Types of .NET Namespace
- 1.5 The Common Language Runtime (CLR)
- 1.6 Assemblies
- 1.7 Answers to Check Your Progress Questions
- 1.8 Summary
- 1.9 Key Words
- 1.10 Self Assessment Questions and Exercises
- 1.11 Further Readings

UNIT 2 BASIC TERMINOLOGIES 19-34

- 2.0 Introduction
- 2.1 Objectives
- 2.2 .Net Component
 - 2.2.1 Selection Objectives of Components
 - 2.2.2 Component Classes
 - 2.2.3 Creating .NET Classes and Components
- 2.3 .Net Garbage Collection
- 2.4 Answers to Check Your Progress Questions
- 2.5 Summary
- 2.6 Key Words
- 2.7 Self Assessment Questions and Exercises
- 2.8 Further Readings

UNIT 3 OOP's CONCEPT 35-72

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Introduction to OOPs
- 3.3 Classes and Objects
- 3.4 Constructor and Destructor
- 3.5 Inheritance
 - 3.5.1 Types of Inheritance
 - 3.5.2 Implementation of Basic Inheritance
- 3.6 Polymorphism
 - 3.6.1 Overloading
 - 3.6.2 Overriding Methods and Properties

- 3.7 Shadowing
- 3.8 Answers to Check Your Progress Questions
- 3.9 Summary
- 3.10 Key Words
- 3.11 Self Assessment Questions and Exercises
- 3.12 Further Readings

BLOCK II: VISUAL BASIC .NET

UNIT 4 INTRODUCTION TO VB.NET

73-100

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Structure and Programming Elements of VB.NET
 - 4.2.1 Keywords
 - 4.2.2 Data Types
 - 4.2.3 Variables
 - 4.2.4 Constants
 - 4.2.5 Operators
- 4.3 Arrays
- 4.4 String Handling
 - 4.4.1 Copy and Concatenating Strings
 - 4.4.2 Adding, Removing and Replacing Strings
 - 4.4.3 Formatting Strings
- 4.5 Answers to Check Your Progress Questions
- 4.6 Summary
- 4.7 Key Words
- 4.8 Self Assessment Questions and Exercises
- 4.9 Further Readings

UNIT 5 CONTROL STATEMENTS

101-125

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Conditional and Looping Statements
- 5.3 Procedures in VB.NET
 - 5.3.1 Sub Procedures
 - 5.3.2 Functions
 - 5.3.3 MsgBox() and InputBox() Functions
- 5.4 Answers to Check Your Progress Questions
- 5.5 Summary
- 5.6 Key Words
- 5.7 Self Assessment Questions and Exercises
- 5.8 Further Readings

UNIT 6 WINDOWS FORMS

126-156

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Windows Forms
 - 6.2.1 MDI Forms
- 6.3 Event

- 6.4 MSGBOX
 - 6.4.1 Simple MessageBox
 - 6.4.2 MessageBox with Title
 - 6.4.3 MessageBox with Buttons
 - 6.4.4 MessageBox with Icon
 - 6.4.5 MessageBox with Default Button
 - 6.4.6 MessageBox with Message Options
 - 6.4.7 MessageBox with Help Button
- 6.5 InputBox()
- 6.6 DialogBox
- 6.7 RichTextBox
- 6.8 Label Class
- 6.9 LinkLabel Control
- 6.10 Passing Forms
- 6.11 Answers to Check Your Progress Questions
- 6.12 Summary
- 6.13 Key Words
- 6.14 Self Assessment Questions and Exercises
- 6.15 Further Readings

BLOCK III: WINDOWS CONTROLS

UNIT 7 INTRODUCTION TO WINDOW CONTROLS

157-181

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Commonly Used Controls
 - 7.2.1 Label (A) Control
 - 7.2.2 TextBox (abl) Control
 - 7.2.3 Button (ab) Control
 - 7.2.4 RadioButton (o) Control
 - 7.2.5 CheckBox (✓) Control
 - 7.2.6 ListBox (l) Control
 - 7.2.7 ComboBox (c) Control
 - 7.2.8 CheckedListBox (cl) Control
 - 7.2.9 HScrollBar (↔) and VScrollBar (↑) Controls
 - 7.2.10 DateTimePicker (dp) Control
 - 7.2.11 Timer (t) Control
 - 7.2.12 Panel
 - 7.2.13 Splitters
 - 7.2.14 Track Bars
 - 7.2.15 Notify Icons
- 7.3 Menu Controls
- 7.4 Answers to Check Your Progress Questions
- 7.5 Summary
- 7.6 Key Words
- 7.7 Self Assessment Questions and Exercises
- 7.8 Further Readings

UNIT 8 TREE AND LISTVIEW **182-203**

- 8.0 Introduction
- 8.1 Objectives
- 8.2 The ListView and TreeView Classes
 - 8.2.1 Toolbar
 - 8.2.2 Status Bar Control
 - 8.2.3 Progress Bar Control
- 8.3 Answers to Check Your Progress Questions
- 8.4 Summary
- 8.5 Key Words
- 8.6 Self Assessment Questions and Exercises
- 8.7 Further Readings

UNIT 9 DEBUGGING AND ERROR HANDLING **204-222**

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Errors
 - 9.2.1 Error Checking Versus Exception Handling
 - 9.2.2 Types of Errors
 - 9.2.3 Error Handling
- 9.3 Exception Handling
 - 9.3.1 Structured Exception Handling
 - 9.3.2 Finally Block
 - 9.3.3 Unstructured Exception Handling on Error Statement
- 9.4 Answers to Check Your Progress Questions
- 9.5 Summary
- 9.6 Key Words
- 9.7 Self Assessment Questions and Exercises
- 9.8 Further Readings

BLOCK IV: ASP.NET**UNIT 10 INTRODUCTION TO ASP.NET** **223-245**

- 10.0 Introduction
- 10.1 Objectives
- 10.2 File Types in ASP.NET
- 10.3 Page Class
- 10.4 HttpRequest
- 10.5 HttpResponse Class
- 10.6 Server Utility
- 10.7 Answers to Check Your Progress Questions
- 10.8 Summary
- 10.9 Key Words
- 10.10 Self Assessment Questions and Exercises
- 10.11 Further Readings

UNIT 11 BASIC WEB CONTROLS	246-301
11.0 Introduction	
11.1 Objectives	
11.2 Web Controls	
11.2.1 Properties of the Server Controls	
11.2.2 Methods of the Server Controls	
11.3 ListControl Class	
11.4 Validation Controls	
11.5 Rich Controls	
11.6 Data Bound Controls	
11.7 Custom Controls	
11.8 Answers to Check Your Progress Questions	
11.9 Summary	
11.10 Key Words	
11.11 Self Assessment Questions and Exercises	
11.12 Further Readings	
UNIT 12 OVERVIEW OF AJAX	302-325
12.0 Introduction	
12.1 Objectives	
12.2 The AJAX Vision	
12.3 Server-side AJAX versus Client-side AJAX	
12.4 The UpdatePanel Control	
12.5 The Timer Control	
12.6 The UpdateProgress Control	
12.7 The ASP.NET AJAX Control Toolkit	
12.8 Answers to Check Your Progress Questions	
12.9 Summary	
12.10 Key Words	
12.11 Self Assessment Questions and Exercises	
12.12 Further Readings	
BLOCK V: ADO.NET	
UNIT 13 INTRODUCTION TO DATABASE ACCESS	326-344
13.0 Introduction	
13.1 Objectives	
13.2 Database Access in the Internet World	
13.3 ADO.NET: An Overview	
13.4 The Connection Class	
13.5 The Command and DataReader Classes	
13.5.1 The Command Class	
13.5.2 The DataReader Class	
13.6 Answers to Check Your Progress Questions	
13.7 Summary	
13.8 Key Words	
13.9 Self Assessment Questions and Exercises	
13.10 Further Readings	

UNIT 14 SQL BASICS

345-366

- 14.0 Introduction
- 14.1 Objectives
- 14.2 Data Components and the Dataset
- 14.3 Disconnected Data
- 14.4 The DataSet Class
- 14.5 The DataTable Class
- 14.6 The DataRow Class
- 14.7 DataColumn Class
- 14.8 Answers to Check Your Progress Questions
- 14.9 Summary
- 14.10 Key Words
- 14.11 Self Assessment Questions and Exercises
- 14.12 Further Readings

INTRODUCTION

NOTES

AMicrosoft has revolutionized the software market by providing many leading software applications. Many object-oriented languages have also been developed by Microsoft for software applications. To bridge the gap between these languages, Microsoft has introduced a framework known as .NET framework that primarily runs on Microsoft Windows. This framework supports several object-oriented languages, such as Visual Basic, C#, Visual C++, etc. The .NET Framework is intended to be used by most new applications created for the Windows platform.

.NET includes two main components: Common Language Runtime (CLR) and class library. CLR is a software environment that is used for executing programs written for .NET framework. The .NET framework's base class library offers user interface, data access, database connectivity, cryptography, network communications and Web application development. Programmers develop software's by integrating their own source code with .NET framework and other class libraries. The .NET framework integrates the business logic of an application developed in different programming languages and services. It has provided major improvements in code reusability, resource management and code specialization, developments of applications, security, deployment and administration of programs developed in various programming languages.

This book, *.NET Programming*, is aimed at providing the readers with basic knowledge. The book follows the Self-Instructional Mode or SIM format wherein each unit begins with an 'Introduction' to the topic of the unit followed by an outline of the 'Objectives'. The detailed content is then presented in a simple and structured form interspersed with 'Check Your Progress' questions to facilitate a better understanding of the topics discussed. The 'Key Words' help the student revise what he/she has learnt. A 'Summary' along with a set of 'Self Assessment Questions and Exercises' is also provided at the end of each unit for effective recapitulation.

BLOCK - I

.NET FRAMEWORKS

*Introduction to
.Net Framework*

UNIT 1 INTRODUCTION TO .NET FRAMEWORK

NOTES

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Evolution of Web Development
 - 1.2.1 ASP.NET: An Overview
 - 1.2.2 Platform Requirements
- 1.3 The Microsoft .NET Framework
- 1.4 .NET Framework Class Library
 - 1.4.1 Types of .NET Namespace
- 1.5 The Common Language Runtime (CLR)
- 1.6 Assemblies
- 1.7 Answers to Check Your Progress Questions
- 1.8 Summary
- 1.9 Key Words
- 1.10 Self Assessment Questions and Exercises
- 1.11 Further Readings

1.0 INTRODUCTION

.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability across several programming languages. Programs written for .NET Framework execute in a software environment, known as Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management and exception handling. FCL and CLR together constitute .NET Framework. You will also learn about the significance of ASP.NET in .NET Framework. ASP.NET is the major part of the Microsoft's .NET Framework. ASP.NET enables hosting Web applications and Web services, with almost any feature from the .NET class library. It also includes a set of Web specific services.

1.1 OBJECTIVES

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

NOTES

- Discuss the evolution of Web development
- Explain the significance of ASP.NET, a major part of Microsoft's .NET Framework
- Discuss the technologies that constitute the .NET Framework
- Understand the need of .NET Framework Class Library (FCL)
- Describe the standardized and non-standardized namespaces provided by .NET FCL
- Discuss the types of assemblies

1.2 EVOLUTION OF WEB DEVELOPMENT

Earlier, Microsoft developed desktop applications using Visual Basic. Visual Basic was very easy to handle. The much debated issue came, when in the mid-1990s, the Internet entered the scene. Microsoft could not move Visual Basic for application development model based on the Internet. Using Internet-based applications means that changes made in one case was immediately made available to each and every user visiting the application through the browser.

Tim Berners-Lee performed the first transmission across HTTP (HyperText Transfer Protocol). Since then, HTTP has become exponentially more popular. When HTTP was first established, developers faced the challenge of designing applications that could discover and interact with each other. To help meet these challenges, standards, such as HTML (HyperText Markup Language) and XML (eXtensible Markup Language) were created. These standards guaranteed that the Web could be used by anyone, located anywhere, using any type of computing system.

At the same time, software vendors faced their own challenges. They needed to develop not only language and programming tools that could integrate with the Web, but also entire frameworks that would allow developers to architect, develop and deploy these applications easily. Major software vendors including IBM, Sun Microsystems and Microsoft rushed to meet this need with a host of products.

ASP.NET 1.0 opened a new chapter in this ongoing arms race. With .NET, Microsoft created an integrated suite of components that combines the building blocks of the Web markup languages and HTTP.

1.2.1 ASP.NET: An Overview

ASP.NET is the major part of the Microsoft's .NET Framework. It is very different from the classic ASP in performance and features. ASP (Active Server Pages), a server side scripting engine, was Microsoft's first development framework for implementing Web applications with dynamic and interactive Web pages. However, the classic ASP had some limitations. It had no support for object orientation and ASP pages were interpreted and not compiled, resulting in poor performance. ASP.NET, released in 2002, is an improved technology that supports .NET's code behind model, better scalability, faster development and support for object orientation. ASP.NET is an entirely new technology for server-side scripting. The following facts about ASP.NET are worth noting.

- **ASP.NET is Integrated with the .NET Framework**

Microsoft has put in great efforts to burgeon their new product, ASP.NET that runs on .NET Framework. The .NET Framework, by definition, is the infrastructure for Microsoft .NET Platform. The .NET Framework provides a unique environment that can be used for developing, implementing and executing various Web services independent of the platform. The .NET Framework supports C++, Visual Basic, JScript, COBOL, Perl, C#, Python and various other languages. Microsoft .NET Framework helped programming made easier and faster, less code, declarative programming model, richer server control hierarchy with events, larger class library and better support for development tools.

- **ASP.NET is Compiled, Not Interpreted**

ASP.NET is a complied language, whereas ASP is an interpreted scripting language. It was written from scratch and is not compatible with the classic ASP. ASP.NET has many new features as compared to the classic ASP. It supports programmable controls, event-driven programming, better language support, XML-based components, user authentication, with accounts and roles, higher scalability, increased performance, compiled code, easier configuration and deployment.

For developers, .NET contains two important parts: the Common Language Runtime (CLR) and the .NET Framework classes. The CLR.NET-compatible compilers translate high-level source code into a special intermediate language, Microsoft Intermediate Language (MSIL). The CLR then takes this Intermediate Language (IL) and converts it into machine-specific instructions. The originality of the .NET system is that various languages may be compiled to MSIL and share their classes. Figure 1.1 shows the compilation in an ASP.NET Web page.

NOTES

NOTES

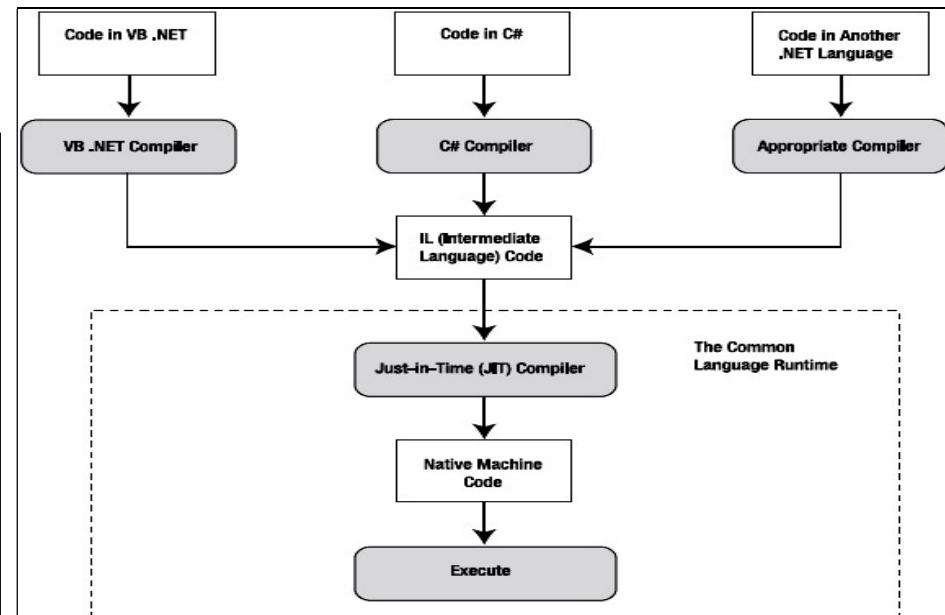


Fig. 1.1 Compilation in an ASP.NET Web Page

- **ASP.NET is Multilingual**

Though you will probably opt to use one language over another when you develop an application, but the choice may not matter because no matter what language you use, the code is compiled in IL. In ASP.NET you are not limited to scripting languages. ASP.NET enables Web developers to create data-driven Websites. The .NET Framework supports C++, Visual Basic, JScript, COBOL, Perl, C#, Python and various other languages. All these languages are used in the development of multi-platform independent .NET Applications.

- **ASP.NET Runs Inside the CLR**

Perhaps, the most important aspect of ASP.NET is that it runs inside the runtime engine of the CLR. The whole of the .NET Framework—that is, all namespaces, applications and classes—are referred to as managed code.

Some of the benefits of CLR are: Automatic Memory Management and Garbage Collection, Type Safety, Extensible Metadata, Structured Error Handling and Multithreading.

ASP.NET aims for performance benefits over other script-based technologies. It simplifies developer's transition from Windows to Web development. It enables us to develop Web applications that include dynamic and data-driven browser-based applications, form-based applications, console applications, and component libraries and Web services. Figure 1.2 shows ASP.NET being run in CLR.

NOTES

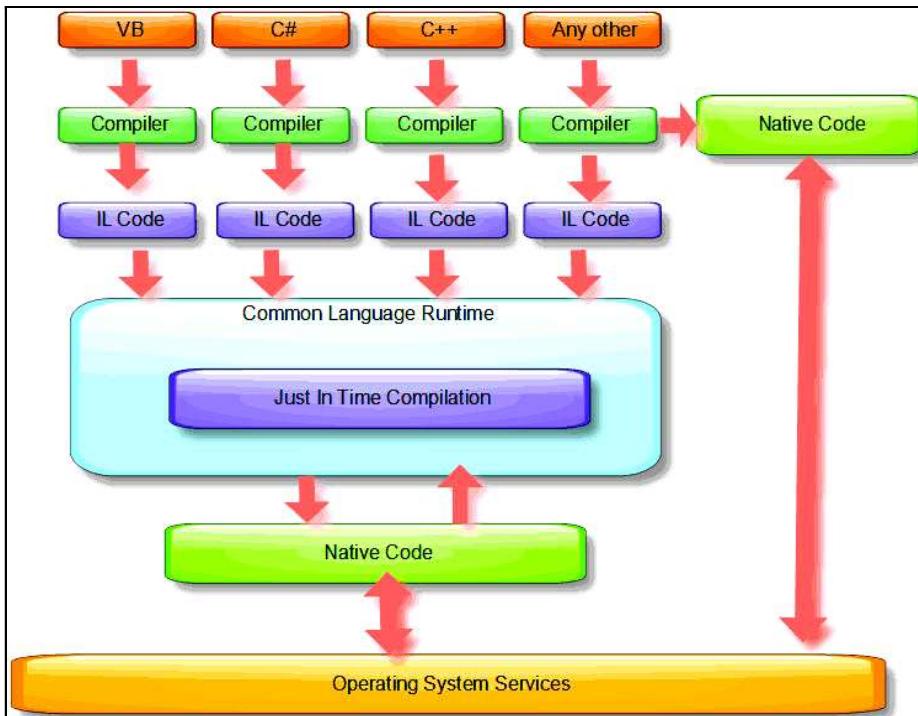


Fig. 1.2 ASP.NET in CLR

- **ASP.NET is Object Oriented**

ASP provides a relatively weak object model. It provides a small set of objects that are treated as variant data types. Variant data types are weakly typed. They require larger amounts of memory, are late-bound, and result in slower performance. Additionally, the compiler and development tools cannot identify them at design time. On the other hand, ASP.NET is truly object oriented. Not only does your code have full access to all objects in the .NET Framework, but you can also exploit all the conventions of an OOP (Object Oriented Programming) environment. One of the best examples of object oriented thinking in ASP.NET is found in server-based controls.

Server-based controls are the essence of encapsulation. Developers can manipulate control objects programmatically using code to customize their appearance, provide data to display and even react to events.

- **ASP.NET is Easy to Deploy and Configure**

One of the biggest problems a Web developer faces during a development cycle is deploying a completed application to a production server. Every installation of the .NET Framework provides the same core classes. As a result, deploying an ASP.NET application is relatively simple. ASP.NET code is easy to understand and use. It includes many features, such as AJAX, which helps to create Web applications, HTML and CSS for presentation, master pages, themes and IntelliSense feature.

NOTES

1.2.2 Platform Requirements

Before you start ASP.NET, it is essential to do some preliminary checks. You must first ensure that you meet the system requirements for installation. When you install Visual Studio .NET, it will also install the MSDN for Visual Studio, which contains valuable information on .NET development. The following are the platform and system requirements:

- **Supported Operating Systems:** Windows 2000 Service Pack 3; Windows 98; Windows 98 Second Edition; Windows ME; Windows Server 2003; Windows XP Service Pack 2.
- **Required Software:**
 - Visual 2010
 - SQL Server 2005 or 2008
- **Disk Space Requirements:** 280 MB (x86), 610 MB (x64)

ASP.NET is supported on Windows 2000 (Professional, Server and Advanced Server), Windows XP Professional, and the Windows Server 2003 family for both client and server applications. In addition, to develop ASP.NET server applications, the following software is also required:

 - Windows 2000 Server or Advanced Server with Service Pack 2, Windows XP Professional or 64-Bit Edition, or one of the Windows Server 2003 family products.
 - MDAC 2.7 for Data.
 - Internet Information Services (IIS).

When you install Visual Studio .NET on a computer running Windows XP Professional or Windows 2000 Server, the .NET Framework and ASP.NET are automatically installed. If you want to install ASP.NET and the .NET Framework by themselves, you can download them over the Web and install them on your server. The following procedure provides instructions on how to do this.

- **To Download and Install ASP.NET on a Computer Running Windows XP Professional or Windows 2000 Server**
 1. Install and start IIS.
 2. At <http://msdn.microsoft.com/downloads/default.asp>, expand **Software Development Kits**, click on **Microsoft .NET Framework SDK**, and then read the instructions, and options for downloading the SDK.
 3. Click the download option that you want, and then click **Yes**.
 4. In the **File Download** dialog box, click **Save**, choose the folder where you want the setup program and Readme files to be downloaded, and then click **Save**.
 5. Review the Readme file for any last minute instructions.

6. In the folder where you downloaded it, double-click the .NET Framework setup program, **setup.exe**.

*Introduction to
.Net Framework*

Table 1.1 lists the hardware required for downloading and installing ASP.NET.

Table 1.1: Hardware Requirements

NOTES

Scenario	Required Processor	Recommended Processor	Required RAM	Recommended RAM
Client	Pentium 90 MHz*	Pentium 90 MHz or Faster	32 MB*	96 MB or Higher
Server	Pentium 133 MHz*	Pentium 133 MHz or Faster	128 MB*	256 MB or Higher

1.3 THE MICROSOFT .NET FRAMEWORK

The .NET Framework, as defined earlier, is the infrastructure for Microsoft .NET Platform. Microsoft started the development of .NET Framework in late 1990s initially under the name of the Next Generation of Windows Services (NGSM). In late 2000, the first beta .NET 1.0 was released. Based on the feedback received from the community many changes were made and .NET Framework version 2.0 was released. It included ASP.NET, ADO.NET for connectivity and Win Forms. Later on, version 3.0 of .NET Framework was released which included Windows Vista and Server 2008 along with technologies, such as WPF, WCF, WF and Card Space. On 12 April 2010, .NET Framework 4.0 was released alongside Visual Studio 2010 which included entity framework ADO.NET and LINQ.

The .NET Framework provides a unique environment that can be used for developing, implementing and executing various Web services independent of the platform. The .NET Framework is really a cluster of several technologies:

- **.NET Languages:** These include C# and VB .NET (Visual Basic .NET), JScript .NET, J# and C++ with managed extensions.
- **CLR (Common Language Runtime):** The CLR is responsible for executing all .NET programs, besides enabling automatic services for these applications. These services include security checking, memory management and optimization.
- **.NET Framework Class Library:** It collects numerous pieces of pre-built functionalities that can be appended to the various applications. Examples of such functionalities include technology sets, such as ADO.NET and Windows Forms. ADO.NET enables you to create database applications and Windows Forms can be used for creating desktop user interfaces.
- **ASP.NET:** It enables hosting Web applications and Web services, with almost any feature from the .NET class library. It also includes a set of Web specific services.

NOTES

- **Visual Studio:** It is a development tool that is characterized by numerous productivity and debugging features.

All these languages are used in the development of multi-platform independent .NET Applications. Microsoft .NET Framework helped programming made easier and faster, less code, declarative programming model, richer server control hierarchy with events, larger class library and better support for development tools.

The .NET Framework consists of three main parts: Programming Languages, Server and Client Technologies and Development Environments.

Programming languages include C#, J# and VB.NET. ASP.NET and Windows Forms are the server and client technologies, respectively. Using Visual Studio, the development tool from Microsoft, Web developers can develop very compelling applications using ASP.NET with the ease of drag-and-drop server controls.

Figure 1.3 shows the .NET Framework architecture.

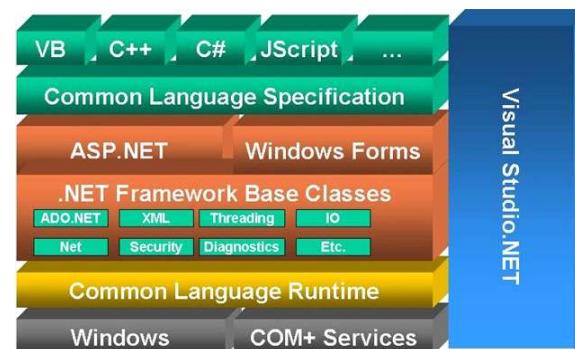


Fig. 1.3 .NET Framework Architecture

At the base is the operating system. CLR lies on the operating system. It is considered as the heart of the .NET Framework. .NET applications are compiled to a common language known as Microsoft Intermediate Language or (MSIL). The CLR, then, handles the compiling of the (MSIL) to machine language, at which point the program is executed.

The CLR environment is also referred to as a managed environment, in which common services, such as garbage collection and security, are automatically provided. The .NET Framework base classes include ADO.NET, XML, IO, NET, security, and so on. Above all are the languages on which the user works or the users interface. Basic programming languages are VB.NET, C#, JScript, etc. Other .NET languages are APL, Eiffel, FORTRAN, Haskell, Mercury, Perl, Python, Pascal, Oberon, etc.

Cross Language Integration

Choosing a programming language depends on your language and the scope of the application you are building. While small applications are often created using

only one language, it is not uncommon to develop large applications using multiple languages.

For client-server applications, you would probably choose a single language but for new enterprise applications, where a large team of developers create components and services for deployment across multiple remote sites, the best choice might be to use several languages depending on developer skills and long-term maintenance expectations. The .NET Platform programming languages include Visual Basic .NET, Visual C#, Managed Extensions for C++, and many other programming languages from various vendors that use .NET Framework services and features through a common set of unified classes.

Microsoft .NET makes it possible to develop applications that are interoperable. This means you can use C# to call the methods and properties of a library written in VB.NET or access C# libraries from VB.NET. This is done through programs that use CLR-compliant types which are not specific to any language.

The Common Type System

The Common Type System (CTS) is a standard that defines a set of types and rules that are common to all languages targeted at the CLR. It is intended to allow programs written in different programming languages to easily share information. It specifies how type definitions and specific values of types are represented in computer memory.

In programming languages, a type can be described as a definition of a set of values, and the allowable operations on those values.

Functions of CTS

The following are the functions of common type system:

- To establish a framework that helps enable cross-language integration, type safety and high performance code execution.
- To provide an object oriented model that supports the complete implementation of many programming languages.
- The CTS also defines the rules that ensure that the data types of objects written in various languages are able to interact with each other.
- The CTS also specifies the rules for type visibility and access to the members of a type.
- It is used to communicate with other languages.

It supports two general category types, i.e., Value Type and Reference type. Value types directly contain their data and are allocated on the stack. Value types can be built-in (implemented by the runtime), user-defined or enumerations.

Reference types store a reference to the value's memory address and are allocated on the heap. Reference types can be self-describing types, pointer types or interface types.

NOTES

Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types and delegates. Converting value types to reference types is also known as boxing.

NOTES

1.4 .NET FRAMEWORK CLASS LIBRARY

The .NET Framework Class Library (FCL) is a collection of reusable classes, interfaces and data types included in the .NET Framework to provide access to system functionality. The .NET FCL acts as the base on which applications, controls and components are built in .NET. It can be used for developing applications, such as console applications, Windows GUI applications, ASP.NET applications, Windows and Web services, workflow-enabled applications, service oriented applications using Windows Communication, XML Web services, etc. FCL acts as a standard library, which can be used in a consistent manner by the .NET languages and common language compliant (CLC-compliant) compilers. The .NET FCL is the key component of .NET framework. It provides core functionalities of .NET architecture, which include base data types, object type, implementation of data structures, garbage collection, security, data access and database connectivity, network communications and support for implementing rich client GUI for both Windows and Web-based applications.

FCL is designed to provide services similar to the Windows Application Programming Interface (API). FCL has its code base as managed, object-oriented and easy to use, while Windows API is unmanaged, modular and cumbersome to use. The .NET FCL is integrated with the Common Language Runtime (CLR) of the Framework, which manages the code execution. Its classes follow the object model as used by the Intermediate Language (IL) and are based on single inheritance. The classes and interfaces are grouped into namespaces so that they can be accessed easily.

Namespaces represent a hierarchy of the defined types formed by a logical group of related classes and interfaces, which can be used by any language targeting the .NET framework. They reside in assemblies, which are deployable units containing details about classes, interfaces and structures. The first part up to the last dot of the full name of a type indicates the namespace, while the last part specifies the type name. This way of using namespaces avoids a naming conflict, which can arise if two class names are same. While System is the root namespace for fundamental types in .NET framework, Object forms the root for all objects.

The classes and interfaces provide an option to use the functionality through implementation (in a concrete class considering it as a base) or only the signatures of methods defined in interface or abstract classes. When using Visual Studio for development of an application, the most common base classes are already referenced in the project, while the types not defined, such as user-defined types in a separate dynamic link library have to be added explicitly so they can be used. The class servicing the needed functionality can be used in code by including an

import directive for the namespace containing the class. Microsoft has also provided guidelines necessary to be adopted for library development, which extend and interact with the .NET Framework. These guidelines cover naming types and members in class libraries, using static and abstract classes, interfaces, members of type, exceptions, etc. Improper usage of the FCL library can adversely affect developer productivity and discourage its usage. FCL is similar to Java Foundation Classes. The main challenge in using FCL is to know the specific class that can provide the required functionality.

The Base Class Library (BCL) or the .NET Framework Class Library (FCL) is a standard library available to all languages. BCL is a set of managed classes that provides many services to the CLR. .NET includes BCL in order to encapsulate a large number of functions, such as file handling, remoting, file reading and writing, Web services and data access.

1.4.1 Types of .NET Namespace

The .NET FCL includes some standardized and non-standardized namespaces as shown in Table 1.2 and Table 1.3.

Table 1.2 Standard Namespaces

Namespace	Description
System	It includes core needs for programming. It has base types, such as String, Date, Time, Boolean, etc.
System.Collections	It defines collections, such as lists, queues, stack, hashtables, etc.
System.IO	It allows you to read from and write to different streams.
System.Security	It allows you to build security into your application based on policy and permissions.
System.Net	It provides interface for many protocols, such as HTTP, FTP and SMTP.
System.Runtime	It allows you to manage the runtime behavior of an application or the CLR.
System.Text	It supports various encodings, regular expressions and a more efficient mechanism for manipulating strings
System.Threading	Helps facilitate multithreaded programming. It allows the synchronizing of "thread activities and access to data" and provides "a pool of system supplied threads."

Table 1.3 Non-Standard Namespaces

Namespace	Description
System.Configuration	It helps to configure data.
System.Data	It represents ADO.NET architecture and is used to access data and data services.
System.Web	It provides Web related functionality.
System.Drawing	It provides access to graphics functionality.
System.ComponentModel	Provides the ability to implement the run-time and design-time behavior of components and controls.
System.Deployment	Allows you to customize the way your application upgrades.
System.Media	Provides you the ability to play system sounds and .wav files.
System.Timers	Allows you to raise an event on a specified interval.
System.Xml	Provides standards-based support for processing XML.
System.Transactions	Provides support for local or distributed transactions.

NOTES

NOTES

Check Your Progress

1. Define the term CTS.
2. Who performed the first transmission across HTTP?
3. When was ASP.NET released?
4. Is ASP.NET object oriented?
5. Name three .NET languages.

1.5 THE COMMON LANGUAGE RUNTIME (CLR)

The (Common Language Runtime) CLR is the engine that supports all the .NET languages. Many modern languages use runtimes. These runtimes may provide libraries used by the language, or they may have the additional responsibility of executing the code. The high level programming languages use its corresponding runtime to run the application.

For example, having an installed Visual Basic runtime is a prerequisite for a computer to run an application which was developed using Visual Basic. This is because only those applications that have been developed with Visual Basic can be run by Visual Basic runtime. Applications developed with other programming languages, such as Java would not be able to be run by Visual Basic runtime.

Besides executing code, the CLR also enables a huge group of related services. Some of these include code verification, optimization and object management. The problem of installing different runtimes for different programming languages is prevented in .NET Framework, by using a common language runtime for all the Microsoft .NET languages. Microsoft .NET Common Language Runtime installed on a computer can run any language that is compatible with Microsoft .NET. All .NET code runs inside the CLR, irrespective of whether a Windows application or a Web service is being run. For example, when a client requests an ASP.NET Web page, the ASP.NET service runs inside the CLR environment, executes the code and creates a final HTML page which is then delivered to the client.

The CLR is the managed runtime environment of Microsoft .NET. In the CLR, code is expressed in the form of byte code called the CIL—Common Intermediate Language (previously called as MSIL – Microsoft Intermediate Language). Developers using the CLR write code in C# or VB.NET or any other .NET language. The CLR provides support for memory management, type safety, exception handling, Just-In-Time (JIT) compilation and automated garbage collection, etc.

NOTES

The implications of the CLR are wide ranging:

- **Deep Language Integration:** Developers using the CLR write code in C# or VB.NET or any other .NET language. At compile time, a .NET compiler converts such code into CIL code. At runtime, the CLR's JIT (Just-In-Time compiler) converts the CIL code into code native to the operating system.
- **Side-By-Side Execution:** The CLR can load many versions of a component simultaneously. This implies that you need not rewrite your applications or replace the current version. Instead, you can upgrade to new versions of ASP.NET.
- **Fewer Errors:** Unlike the lower-level languages, such as C++, the CLR generates little or no errors.

Even though the CLR provides some revolutionary benefits, it has some potential drawbacks. Some of the more common drawbacks are the following:

- **Code Transparency:** It is much easier to disassemble and decipher the code by any programmer.
- **Questionable Cross-Platform Support:** As .NET is characterized by many features and technologies that are platform-specific as well as operating system-specific, hence, .NET will not be as favorable a language, such as Java. Figure 1.4 illustrates the .NET application compilations.

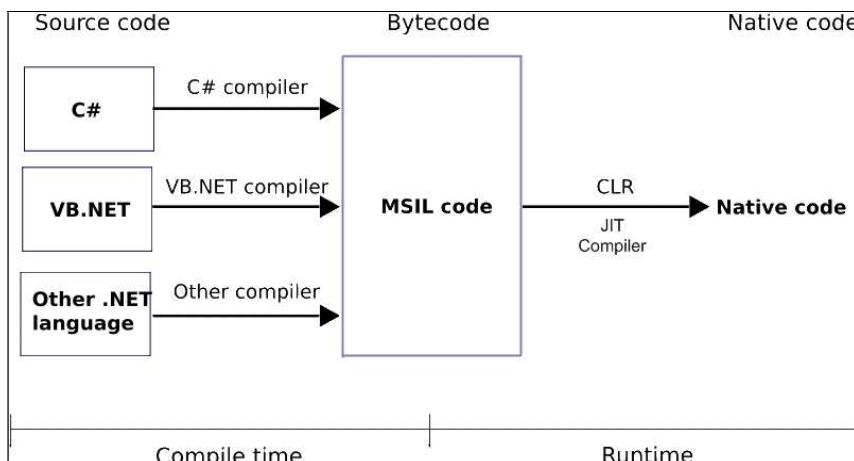


Fig. 1.4 .NET Application Compilations

The Just-in-Time Compiler

The Just-In-Time (JIT) compiler is a component that converts the intermediate machine-independent CIL code to machine-dependent code. The CIL code and its metadata are loaded into the memory by the CLR. The JIT compiler then compiles this CIL code to machine code at runtime.

There is a pair of JIT compilers in Microsoft.NET runtime—namely the standard JIT compiler and the EconoJIT compiler. Even though the EconoJIT

compiler compiles quicker than the standard JIT compiler, the code generated is not as optimized as the code generated by the standard JIT compiler. Figure 1.5 illustrates the compilation and execution process.

NOTES

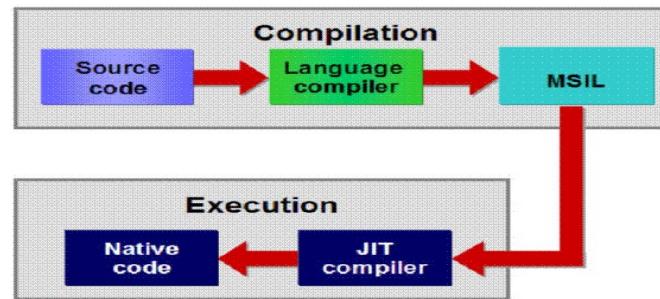


Fig. 1.5 Compilation and Execution Process

Advantages of JIT Compiler

The following are the advantages of JIT compiler.

1. The JIT enables the code to be optimized to a particular operating system.
2. This gives portability to the code.
3. It is also possible to convert the source code to the native code directly by bypassing the tasks done by the JIT. This helps to reduce the load in the JIT.

1.6 ASSEMBLIES

An assembly is a logical grouping of functionalities in a physical file. An assembly in ASP.NET is a collection of single files or multiple files.

The following are the characteristic features of assemblies:

- Assemblies are the building blocks of .NET Framework.
- They exist physically as a DLL or an EXE file.
- One assembly can contain one or more files.
- The constituent files can include any file type, such as image files and text files, along with DLLs or EXEs.
- When you compile your source code by default, the EXE/DLL generated is actually an assembly.
- Every assembly file contains information about itself. This information is called as Assembly Manifest.
- The Assembly Manifest contains data about assembly's identity (the author name, version requirements of the assembly), culture information, type, dependencies, the security requirements and the various files that form part of the assembly.

- The biggest advantage of using assemblies is that developers can create applications without interfering with other applications on the system.

Note: The assembly used for one application is not applied to another application. However, one assembly can be shared with other applications. In this case, the assembly has to be placed in the ‘Bin’ directory of the application that uses it.

Thus, you can create two types of assemblies in ASP.NET:

1. Private Assembly
2. Shared Assembly

Private and Shared Assemblies

The assembly which is used only by a single application is called a private assembly. Private ASP.NET assemblies are created when you build component files like DLLs that can be applied to one application. Suppose you have created a DLL which encapsulates your business logic. This DLL will be used by your client application only and not by any other application. Thus, the assembly is private to your application.

Shared ASP.NET assemblies are created when you want to share the component files across multiple applications. Suppose that you are creating a general purpose DLL which provides functionality which will be used by a variety of applications. Now, instead of each client application having its own copy of DLL you can place the DLL in ‘Global Assembly Cache (GAC)’. Such assemblies are called as shared assemblies.

The Global Assembly Cache (GAC)

- Global assembly cache is a special disk folder where all the shared assemblies will be kept. It is located under <drive>:\WinNT\Assembly folder.
- It is the central storage location for shared assemblies.
- An assembly placed in GAC must have **global unique identifier** called as **strong names**.
- Strong names consist of the assembly’s identity information, a public key and a digital signature.

Static and Dynamic Assemblies

- Static assemblies are stored on the disk as physical files.
- Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (Bitmaps, JPEG files, Resource files, and so on).
- Dynamic assemblies are not saved to the disk before execution and are executed directly from the memory.

NOTES

NOTES

Assembly Version

An assembly consists of a 128-bit version number that is divided into four distinct parts: the major version, the minor version, the build version and the revision (release) version.

For example, 1.0.1.1 implies that the assembly has 1 as the major version, 0 as the minor version, 1 as the build version and 1 as the release version.

Check Your Progress

6. What is .NET Framework Class Library?
7. What are the types of assemblies?

1.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The Common Type System (CTS) is a standard that defines a set of types and rules that are common to all languages targeted at the CLR. It is intended to allow programs written in different programming languages to easily share information.
2. Tim Berners-Lee performed the first transmission across HTTP (HyperText Transfer Protocol). Since then, HTTP has become exponentially more popular.
3. ASP.NET was released in 2002 and is an improved technology that supports .NET's code behind model, better scalability, faster development and support for object orientation.
4. ASP.NET is truly object oriented. Not only does your code have full access to all objects in the .NET Framework, but you can also exploit all the conventions of an OOP (Object Oriented Programming) environment.
5. .NET supports many languages; some of them are C#, VB.NET and J#.
6. The .NET Framework Class Library (FCL) is a collection of reusable classes, interfaces and data types included in the .NET Framework to provide access to system functionality.
7. There are two types of assemblies in ASP.Net i.e. private and shared.

1.8 SUMMARY

- ASP.NET is the major part of the Microsoft's .NET Framework. It is very different from the classic ASP in performance and features. ASP (Active Server Pages), a server side scripting engine, was Microsoft's first

development framework for implementing Web applications with dynamic and interactive Web pages.

- The .NET Framework provides a unique environment that can be used for developing, implementing and executing various Web services independent of the platform. The .NET Framework supports C++, Visual Basic, JScript, COBOL, Perl, C#, Python and various other languages.
- .NET contains two important parts: the Common Language Runtime (CLR) and the .NET Framework classes.
- ASP.NET is truly object oriented. Not only does your code have full access to all objects in the .NET Framework, but you can also exploit all the conventions of an OOP (Object Oriented Programming) environment.
- The CLR is the engine that supports all the .NET languages. The CLR is the managed runtime environment of Microsoft .NET.
- An assembly is a logical grouping of functionalities in a physical file. An assembly in ASP.NET is a collection of single files or multiple files.

NOTES

1.9 KEY WORDS

- **.NET Framework:** Provides a unique environment that can be used for developing, implementing and executing various Web services independent of the platform.
- **.NET Languages:** Include C# and VB .NET (Visual Basic .NET), JScript .NET, J# and C++ with managed extensions.
- **Common Type System (CTS):** A standard that defines a set of types and rules that are common to all languages targeted at the CLR.
- **Base Class Library (BCL):** A set of managed classes that provides many services to the CLR.

1.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. List any four characteristics of ASP.NET.
2. How does ASP.NET provide increased performance?
3. What are the platform requirements of ASP.NET?
4. List the functions of CTS.
5. Write a short note on assemblies.

NOTES

Long Answer Questions

1. Explain the working of CLR.
2. Explain .NET Framework architecture. Illustrate the process with the help of a diagram.
3. What are assemblies? Explain its versions.
4. What is a namespace? Explain the different types of namespaces.

1.11 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 2 BASIC TERMINOLOGIES

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 .Net Component
 - 2.2.1 Selection Objectives of Components
 - 2.2.2 Component Classes
 - 2.2.3 Creating .NET Classes and Components
- 2.3 .Net Garbage Collection
- 2.4 Answers to Check Your Progress Questions
- 2.5 Summary
- 2.6 Key Words
- 2.7 Self Assessment Questions and Exercises
- 2.8 Further Readings

NOTES

2.0 INTRODUCTION

In this unit, you will learn about the .NET Framework component and garbage collection. .NET Framework component is an object that is reusable, can interact with other objects, and provides control over external resources and design-time support. An important feature of components is that they are designable, which means that a class that is a component can be used in the Visual Studio Integrated Development Environment.

2.1 OBJECTIVES

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

- Describe the basic concept of components
- Understand the selection objectives of components
- Explain how to create .NET classes and components
- Explain how garbage collector works

2.2 .NET COMPONENT

Introduction to Components

A component is a class that implements the `System.ComponentModel.IComponent` interface or that derives directly or indirectly from a class that implements `IComponent`. A .NET Framework component is an object that is reusable, can interact with other objects, and provides control over external

NOTES

resources and design-time support. An important feature of components is that they are designable, which means that a class that is a component can be used in the Visual Studio Integrated Development Environment. A component can be added to the Toolbox, dragged and dropped onto a form, and manipulated on a design surface. Note that base design-time support for components is built into the .NET Framework; a component developer does not have to do any additional work to take advantage of the base design-time functionality. A control is similar to a component, as both are designable. However, a control provides a user interface, while a component does not.

Microsoft .NET applications are built from components. All .NET objects expose important attributes, such as properties, methods and events. These attributes form the foundation of object oriented programming. Typically, simple .NET object oriented programming involves creating a class, adding the properties, methods, and events required by the class, and including the class in various applications. Component based development takes this basic concept to a higher level. Although the components you build in .NET are based on object oriented programming principles, they go beyond the simple classes that you might use in multiple applications.

A component is a special type of executable created from a .NET project. After compilation the component is typically referenced by applications that require the services provided by the component. In many .NET Web environments, components run on the Web server and provide data and other services (such as security, communications, and graphics) to the Web Services operating on the server. In a Windows Form application a .NET component performs the same role as on a Web server, but on a reduced scale.

.NET components provide a programmable interface that is accessed by consumer applications (often called *client applications*). The component interface consists of a number of properties, methods, and events that are exposed by the classes contained within the component. In other words, a component is a compiled set of classes that support the services provided by the component. The classes expose their services through the properties, methods, and events that comprise the component's interface.

Simple .NET object oriented programming involves not much more than creating a class, adding the properties, methods, and events required by the class, and including the class in different applications. A .NET component, however, is a pre-compiled class module with a .DLL (Dynamically Linked Library) extension. At run time, a .NET component is invoked and loaded into memory to be used by some consumer application. These .NET components are most often built and tested as independent .NET projects and are not necessarily part of another project. After compilation into a .NET DLL, these components can be added to many .NET applications as plug-in service providers. Each .NET DLL component may contain multiple classes. This means that each DLL may expose a single class or a

variety of classes. For instance, if you build a .NET DLL that supports remote data access, the DLL might need to expose separate classes for the database and for the variety of DataSets also produced by the DLL. Or, you may have a single DLL that supports error handling in your applications. In this case, the DLL exposes a single `ErrorHandler` class that performs all of the error handling required by consumer applications.

All Microsoft Windows applications run in the computer's memory, and use the computer's resources, such as disk space, networking services and graphics capabilities. Windows is a multi-tasking operating system, which simply means that multiple applications simultaneously share the computer's memory and other resources. Windows manages each application, and the memory required to run that application, as a *process*. The memory occupied by a Windows application and its data is referred to as the application's *process space*. Among the important jobs performed by Windows are making sure that the process spaces occupied by applications do no overlap, and providing memory to each process as needed. Under certain circumstances, Windows may allow process space to be shared among a number of applications. This is particularly true in the case of data or services (such as networking services) that must be shared between applications. Other times the memory occupied by an application is owned solely by the EXE running in that process space.

Many applications such as Microsoft Word and Microsoft Excel support dozens or hundreds of different features. Rather than distribute these application as very large single files, Microsoft has broken the basic functions supported by Word and Excel into multiple executables. The main program ends with an .EXE extension, while most of the other executable portions are files ending in .DLL.

When Word or Excel needs the functionality (perhaps a spell checker, or recalculation engine) contained within a DLL, Windows loads the DLL into the process space occupied by the main application. Windows then manages the main program and the executable code contained within the DLL as a single process. A DLL is called an *in-process* resource because it executes within the process space occupied by another application.

There are other situations where resources are not loaded in-process. For instance, when Word needs to print a document, Word issues a request to Windows to load a particular printer driver. That same printer driver may be needed by other applications, so Windows loads the driver as an *out-of-process* service so that more than one application can access the driver. This also means that Windows does not have to load multiple copies of the driver into memory.

In-Process Components

In .NET, components built as DLLs run within the process space of the host application and share memory and processor time with their host applications. At run time, the component (which is part of the host application's assembly and is

NOTES

NOTES

referenced by its manifest) is loaded from disk and added to the host application's process space. Because no remote procedure calls are generated to mediate communication between the component and its host, setting and reading property values, invoking methods, and responding to events raised by the component occurs very quickly.

Out-of-Process Components

An alternate architecture involves server applications that run as independent processes outside of the client application process space. These server applications usually (but not always) have an EXE file name extension. When Windows loads an out-of-process component, a separate process space is created for the component, and Windows manages the out-of-process component's resource requirements independently of the client application. Windows mediates the dialog between the server application (that is, the component) and the client (the consumer) by passing messages between them.

2.2.1 Selection Objectives of Components

If your class will be used on a design surface (such as the Windows Forms or Web Forms Designer) but has no user interface, it should be a component and implement `IComponent`, or derive from a class that directly or indirectly implements `IComponent`.

The `Component` and `MarshalByValueComponent` classes are base implementations of the `IComponent` interface. The main difference between these classes is that the `Component` class is marshaled by reference, while `IComponent` is marshaled by value. The following list provides broad guidelines for implementing classes for using components:

- If your component needs to be marshaled by reference, derive from `Component`.
- If your component needs to be marshaled by value, derive from `MarshalByValueComponent`.
- If your component cannot derive from one of the base implementations due to single inheritance, implement `IComponent`.

2.2.2 Component Classes

The `System.ComponentModel` namespace provides classes that are used to implement the run-time and design-time behavior of components and controls. This namespace includes the base classes and interfaces for implementing attributes and type converters, binding to data sources, and licensing components. The core component classes are:

- **Component:** A base implementation for the `IComponent` interface. This class enables object sharing between applications.

- `MarshalByValueComponent`: A base implementation for the `IComponent` interface.
- `Container`: The base implementation for the `IContainer` interface. This class encapsulates zero or more components.

Classes commonly used for describing and persisting components are as follows:

- `TypeDescriptor`: Provides information about the characteristics for a component, such as its attributes, properties, and events.
- `EventDescriptor`: Provides information about an event.
- `PropertyDescriptor`: Provides information about a property.

NOTES

2.2.3 Creating .NET Classes and Components

This section demonstrates most of the principles of class and component development in .NET. First, you will build a simple .NET class. Once you have built the class, you will use the same code to build a component and include the component in a small .NET application. A typical requirement of distributed database systems is to synchronize and reconcile updates performed at remote locations. In a .NET-hosted database application, the same record may be updated at multiple sites and submitted to the server for storage. Or, similar records may be created at multiple sites and sent to the server for processing. In such cases, it may be important to know exactly when each record was updated or created so that newer records may be treated differently than older records. In most such cases, the application assigns a timestamp to database records as the data is modified or added. The server can compare the timestamp on one record with the timestamp on another record and process the records accordingly. However, when the data has been updated in different time zones, it is important to use an appropriate base time for the timestamps. This is often done by using the server computer's time as the timestamp rather than the local time for the data entry.

This is an ideal use of a .NET component. The .NET server can expose its time as a Web Service component that can be read and used by any consumer application. Providing the server time component as a Web Service helps reconcile data changes occurring at geographically distant locations. One major difference between Visual Basic 6.0 classes and .NET classes is that each Visual Basic 6.0 class module (file) contains one and only one class. This means that a project exposing a lot of different classes contains a lot of different class modules (files).

In .NET, a class can be created just about anywhere in your code. This means that a file in a .NET application can contain one or many classes. Each class in a .NET code module is defined by a `Public Class...End Class` code block within the module. This architecture makes it easy to segment the classes developed within a .NET project into logical groups.

Creating a Class

Follow these steps to build a simple class module that returns the host computer's date and time:

NOTES

1. Start Visual Studio .NET and open a new Windows Application project.
2. Ignoring the default Windows Form in the project, on the **Project** menu, click **Add Class** to add a new file to the project.
3. Change the name of the generated Public Class from `Class1` to `ServerTime`.
4. Enter the following code to define the `ServerTime` class.

```
Public Class ServerTime
    Private mdtTime As DateTime
    ReadOnly Property TimeStamp() As String
        Get
            mdtTime = Now()
            Return CStr(mdtTime)
        End Get
    End Property
End Class
```

The private variable named `mdtTime` is not entirely necessary in this class; you could simply return the current date and time without first assigning it to the `mdtTime` variable. But, because programming projects so often grow beyond their initial specification, there may be a need in the future to have a module-level variable shared by multiple properties and/or methods within the `ServerTime` class.

Next, use the Windows Form that was automatically created in the new project to test the functioning of the `ServerTime` class. Follow these steps to create an object from the `ServerTime` class and use it in the Windows Form:

1. Rename the form `Form1` to `frmConsumer` to emphasize its role as the consumer of the class module.
2. Add a `TextBox` to the Windows Form and name it as `txtServerTime`.
3. Delete the string in the `Text` property so that it is blank.
4. Add a button to the Windows Form and name it as `btnGetServerTime`.
5. Set the `Text` property of this button to `Get Server Time`.
6. Double-click the button to add the following code behind the Windows Form:

```
Private Sub btnGetServerTime_Click( _ ByVal sender As System.Object, _ ByVal e As System.EventArgs) _ Handles btnGetServerTime.Click
    Dim st As ServerTime
    st = New ServerTime()
```

```
txtServerTime.Text = st.TimeStamp  
End Sub
```

Basic Terminologies

As the `ServerTime` class is included in this project's assembly, there is no need to reference it from the consumer form's code module. .NET is able to locate the `ServerTime` class and instantiate the `st` object variable from the information in the class. When this code has been added to the button's `Click` event procedure, the current date and time are written into `txtServerTime` each time the button is clicked. Press **F5** to run this application. Click **Get Server Time** and, in the text box, you should see a **Date and Time**.

NOTES

Creating the Component

You have created a simple class module that returns the current date and time, and tested the class module in a .NET Windows Form. Follow these steps to build a .NET DLL component:

1. Start Visual Studio .NET and open a new Class Library project. In the **New Project** dialog box, name the project `ServerTime`.
2. Change the name of the class from `Class1` to `ServerTime`.
3. Enter the following code into the new `ServerTime` class module:

```
Public Class ServerTime  
    Private mdtTime As DateTime  
    ReadOnly Property TimeStamp() As String  
        Get  
            mdtTime = Now()  
            Return CStr(mdtTime)  
        End Get  
    End Property  
End Class
```

You will now compile this `ServerTime` class as a DLL by clicking **Build** on the **Debug** menu or by using the **Ctrl+Shift+B** keystroke combination. The DLL that results from the build command is placed into the `\bin` directory immediately below your .NET project directory. By default, the DLL has the same name as your component project. For instance, if you named the project `TimeStamp` in the New Project dialog, the DLL produced by your project will be named `TimeStamp.DLL`. This name is also the default Namespace name for the classes contained within the project.

Now, your project is named as `ServerTime`. You also named the only class within the project `ServerTime`. Therefore, your DLL name will be `ServerTime.DLL` and consumer applications of this DLL will reference `ServerTime.ServerTime` to create an object of this class.

Create a DLL Consumer Application

Once the DLL project has been compiled, its services are available to any Windows Form or WebForm .NET application. In this section, you will build a simple

Windows Form consumer application that uses the `ServerTime` class to retrieve the computer's date and time.

NOTES

Follow these steps to create the consumer application:

1. Start Visual Studio .NET, select Windows Application as the new project type, and name the project `DLLConsumer1`.
2. Set the Name property of the default Windows Form to `frmConsumer`.
3. Add a button control to the default Windows Form and name it `btnGetServerTime`.
4. Add a TextBox to the form and name it `txtServerTime`.

You need to set a reference to the `ServerTime` DLL so that this form will be able to retrieve the components services. Do this by following the steps below.

1. To open the **Add Reference** dialog box as shown in Figure 2.1, on the **Project** menu, click **Add Reference**.

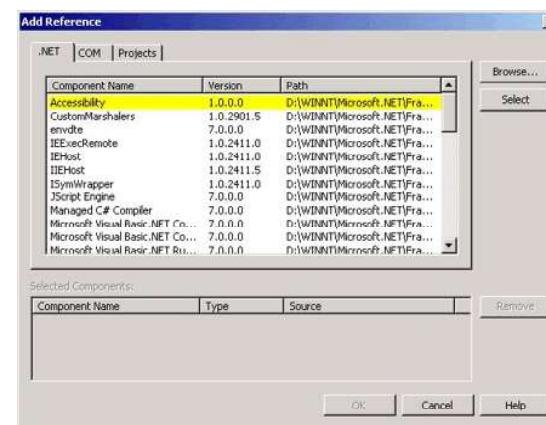


Fig. 2.1 Add Reference Dialog Box

2. Click the **Projects** tab and then click **Browse** to locate the component DLL built in the preceding section (see Figure 2.2).

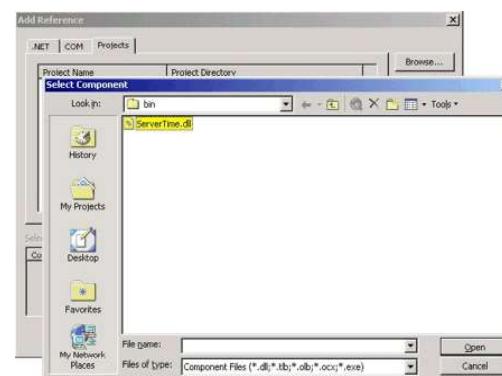


Fig. 2.2 Selecting a DLL Reference

3. Select the ServerTime.DLL file, click **Open**, and then click **OK**.

The Solution Explorer, as shown in Figure 2.3, now shows the ServerTime component added as a reference in your application. This means that all of the classes, along with their properties, methods, and events, are now available to the consumer application.

NOTES

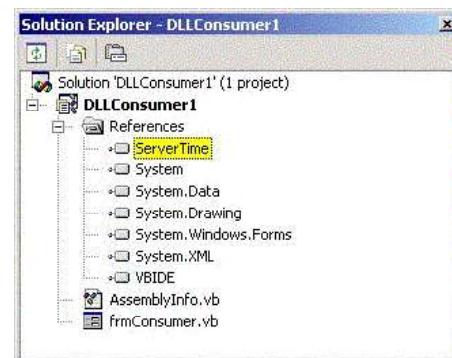


Fig. 2.3 Solution Explorer

Add the following code to the Click event of btnGetServerTime:

```
Private Sub btnGetServerTime_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles btnGetServerTime.Click
    Dim st As ServerTime
    st = New ServerTime()
    txtServerTime.Text = st.TimeStamp
End Sub
```

Notice that this code is identical to the code used earlier to test the operation of the ServerTime class module. In actual practice, a WebForm or Windows Form might use the ServerTime.TimeStamp property to update a TimeStamp field in a new or updated record in a DataSet object. Assuming that the ServerTime DLL was actually running on a server computer (whether LAN- or Web-based) the time returned by the TimeStamp property reflects the actual time on the server. This means that the local computer could be in a different time zone or have its time set incorrectly, yet use the same time base as other computers using the server. An alternate way to write the previous code example is:

```
Private Sub btnGetServerTime_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles btnGetServerTime.Click
    Dim st As New ServerTime
```

NOTES

```

txtServerTime.Text = st.TimeStamp
End Sub

```

In this case, the `st` object variable is declared and instantiated as a single step. Some developers routinely declare and instantiate at the same time to reduce the amount of typing they must do; others prefer to declare an object and instantiate it only when the object is actually needed by the application. In most cases, it makes no difference whether you instantiate earlier or later. In some situations, it makes more sense to instantiate the object only when it is actually required by the application. For instance, consider the `ServerTime` component. As the developer of an application, you may choose to instantiate only after the user adds or updates information in the database, when you will need the correct server time. One of the objectives of distributed applications is to minimize the number of trips to the server. There's no need to bother the server by instantiating the `ServerTime` object until it is actually needed by the consumer application.

Implementing Error Handling

Distributed applications must be very reliable. You cannot allow errors and problems to bring down server-based applications, such as a .NET Web Service. Because .NET DLLs run within the same process space as their host applications, an unhandled exception in a .NET DLL can crash the entire Web Service, disrupting all users working with that Web Service. To make matters worse, server-side crashes can disrupt the database engine's activities, leaving records locked or in indeterminate states. For all these reasons, you must add adequate error handling to all of your .NET components.

The following code shows effective error handling added to the `TimeStamp` property in the `ServerTime` class. Notice the use of the `Try`, `Catch` and `End Try` statements:

```

ReadOnly Property TimeStamp() As String
    Get
        Try
            mdtTime = Now()
            Return CStr(mdtTime)
        Catch e As Exception
            'Return the exception to the
            'consumer for processing:
            Throw e
        End Try
    End Get
End Property

```

The `Try...End Try` statements wrap all of the property's logic and provide the error trap required to catch any errors raised by the property's code. Each property

within the class must have its own Try...End Try block, and should throw an exception back to the consumer application if something goes wrong. It is then up to the consumer application to handle the error. A more sophisticated approach to error handling within the SystemTime component is shown in the following code:

```
ReadOnly Property TimeStamp() As String
    Get
        Try
            mTime = Now()
            Return CStr(mTime)
        Catch e As Exception
            'Return the exception to the
            'consumer for processing:
            Throw e
        End Try
    End Get
End Property
```

In this case, the Catch block returns the Exception object (e) to the consumer application. The Exception object contains information about the exception that has occurred and may be useful to the consumer application.

Visual Basic .NET error handling is much different than the error handling in Visual Basic 6.0. The Visual Basic .NET Try...End Try block is described as ‘structured error handling’ and is a vast improvement over the Visual Basic 6.0 On Error statement. In Visual Basic 6.0, when an error occurred, processing was unconditionally redirected to another part of a procedure. This sometimes made it difficult to exactly determine the path taken by an application when an exception occurred. The general approach to Visual Basic 6.0 exception handling was not rigorously enforced by the VBA compiler, so each programmer was pretty much free to adopt their own approach to handling errors. In Visual Basic .NET, the exception thrown by the component is captured by the Try...End Try block in the consumer application, as shown in the following code:

```
Private Sub btnGetServerTime_Click(
    ByVal sender As System.Object,
    ByVal e As System.EventArgs)
    Handles btnGetServerTime.Click
    Dim st As New ServerTime
    Try
        txtServerTime.Text = st.TimeStamp
    Catch e As Exception
```

NOTES

```
    MessageBox.Show(e.Message)
```

```
End Try
```

```
End Sub
```

NOTES

This is another way that .NET is different from Visual Basic 6.0. The components in a .NET application are tightly bound into the logic of the consumer application, and often you do not have to wait until run time before seeing problems that arise from the integration of consumer applications with their components.

Although Visual Basic.NET supports the obsolete `On Error` statement for backwards compatibility, you should adopt the `Try...End Try` structured error handling in your Visual Basic .NET projects. You will find that adding `Try...End Try` blocks to your Visual Basic .NET code results in a much smoother error handling process than the old `On Error GoTo` statement.

2.3 .NET GARBAGE COLLECTION

Garbage collection is an automatic memory management mechanism in .NET. This mechanism helps to free the memory automatically when it is no longer used by the application. The CLR allocates memory from the heap, for each time a new object is created. Since memory is finite, there is a need to clean up unused objects in order to reclaim memory used by them. The job of garbage collector is to free the unused memory.

How Garbage Collector Works?

Normally, the heap memory is divided into three generations: Generation 0 is for short live objects, Generation 1 is for medium live objects which are moved from Generation 0. Generation 3 is mostly for stable objects.

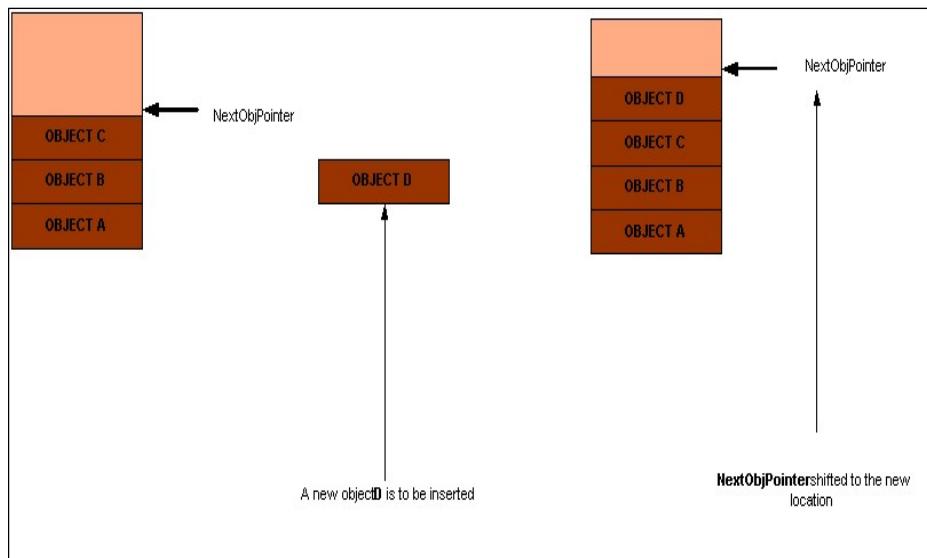
Garbage collector takes into account the following two facts:

1. Newly created objects tend to have short lives.
2. The older an object is, the longer it will survive.

The Garbage Collection Process

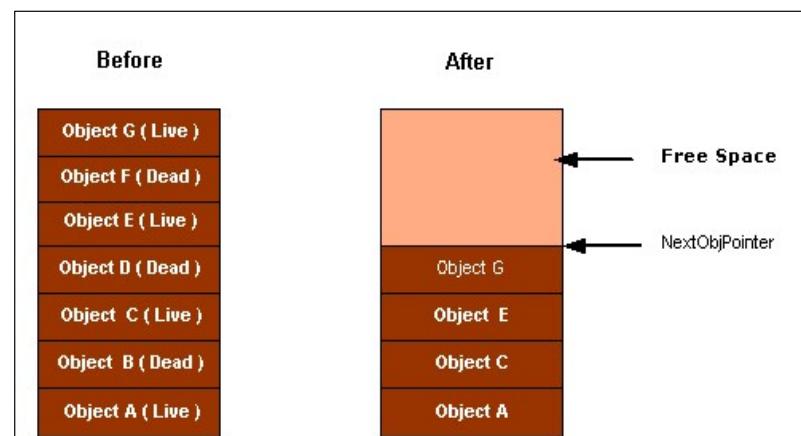
- When an object is created, it is placed in Generation 0. As most objects are short lived, only a small percentage of young objects are likely to survive their first collection.
- Once an object survives the first garbage collection, it gets promoted to Generation 1. All objects in Generation 1 that survive get compacted and are promoted to Generation 2, and so on. Generation 0 then contains no object.

Figure 2.4 illustrate the garbage collection process.

**NOTES****Fig. 2.4 Garbage Collection Process**

- Thus, dividing the heap into generations of objects and collecting and compacting younger generation objects improves the efficiency of the basic underlying garbage collection algorithm by reclaiming a significant amount of space from the heap and also by being faster.

Freeing Memory Regions: Dead objects are removed from the heap to allocate new objects.

**Fig. 2.5 Freeing Memory Regions****Advantages of Garbage Collection**

The advantage of garbage collection is that developers need not worry about the memory management when writing the code, i.e., the garbage collector will free the memory as and when needed. Figure 2.5 illustrates the free memory regions.

Drawbacks of Garbage Collection

- (i) Objects with destructors take up more resources as they stay for a longer period of time in memory even when they are not required.
- (ii) Finalization takes place as a separate thread, again consuming the resources.

Check Your Progress

1. What is a .NET framework component?
2. Name the base implementations of the `IComponent` interface.
3. What do you understand by garbage collection?

2.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A .NET Framework component is an object that is reusable, can interact with other objects, and provides control over external resources and design-time support. An important feature of components is that they are designable, which means that a class that is a component can be used in the Visual Studio Integrated Development Environment.
2. The `Component` and `MarshalByValueComponent` classes are base implementations of the `IComponent` interface. The main difference between these classes is that the `Component` class is marshaled by reference, while `IComponent` is marshaled by value.
3. Garbage collection is an automatic memory management mechanism in .NET. This mechanism helps to free the memory automatically when it is no longer used by the application.

2.5 SUMMARY

- A component is a class that implements the `System.ComponentModel.IComponent` interface or that derives directly or indirectly from a class that implements `IComponent`.
- A .NET Framework component is an object that is reusable, can interact with other objects, and provides control over external resources and design-time support.
- Microsoft .NET applications are built from components. All .NET objects expose important attributes, such as properties, methods and events.
- .NET components provide a programmable interface that is accessed by consumer applications (often called *client applications*). The component

interface consists of a number of properties, methods, and events that are exposed by the classes contained within the component.

- In .NET, components built as DLLs run within the process space of the host application and share memory and processor time with their host applications.
- The `System.ComponentModel` namespace provides classes that are used to implement the run-time and design-time behavior of components and controls.
- In .NET, a class can be created just about anywhere in your code. This means that a file in a .NET application can contain one or many classes. Each class in a .NET code module is defined by a `Public Class...End Class` code block within the module.
- Once the DLL project has been compiled, its services are available to any Windows Form or WebForm .NET application.
- Garbage collection is an automatic memory management mechanism in .NET. This mechanism helps to free the memory automatically when it is no longer used by the application.
- The advantage of garbage collection is that developers need not worry about the memory management when writing the code, i.e., the garbage collector will free the memory as and when needed.

NOTES

2.6 KEY WORDS

- **Garbage Collection:** A mechanism that helps to free the memory automatically when it is no longer used by the application.
- **Component:** A base implementation for the `IComponent` interface. This class enables object sharing between applications.
- **TypeDescriptor:** Provides information about the characteristics for a component, such as its attributes, properties, and events.
- **Container:** The base implementation for the `IContainer` interface. This class encapsulates zero or more components.

2.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Define the term component.
2. State the significance of In-Process Components.
3. What are component classes?

Long Answer Questions

NOTES

1. Describe the process of selection objectives of components and their significance.
2. Discuss the method to create .NET classes and components with the help of examples.
3. Explain how the garbage collector works.

2.8 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 3 OOP's CONCEPT

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Introduction to OOPs
- 3.3 Classes and Objects
- 3.4 Constructor and Destructor
- 3.5 Inheritance
 - 3.5.1 Types of Inheritance
 - 3.5.2 Implementation of Basic Inheritance
- 3.6 Polymorphism
 - 3.6.1 Overloading
 - 3.6.2 Overriding Methods and Properties
- 3.7 Shadowing
- 3.8 Answers to Check Your Progress Questions
- 3.9 Summary
- 3.10 Key Words
- 3.11 Self Assessment Questions and Exercises
- 3.12 Further Readings

NOTES

3.0 INTRODUCTION

Each programming language follows one or the other programming paradigm, which describes the structure of a program. In other words, a programming paradigm determines how the instructions are placed in a program. There is no specific rule to decide which programming paradigm is to be followed, as different paradigms can be used to develop different software. Initially, the programming languages followed **unstructured programming paradigm** in which all the instructions of a program were written one after the other in a single function. This programming approach was suitable for writing only small and simple programs. For large and complex programs, it became difficult to trace and debug errors.

In late 1960s, the high-level languages, such as C and Pascal were developed, which provided a structured way of writing programs. Structured programming (also known as **procedural programming**) was a powerful and an easy approach of writing complex programs. In **procedural programming**, programs are divided into different *procedures* (also known as functions, routines or subroutines) and each procedure contains a set of instructions that performs a specific task. This approach follows the top-down approach for designing the program. That is, first the entire program is divided into a number of subroutines and these subroutines are again divided into small subroutines, and so on until each subroutine becomes an indivisible unit.

NOTES

Though writing programs using procedural programming paradigm is a simple and easy task, there are some limitations also. In procedural programming approach, the emphasis is on the functionality of the software rather than the data used by the functions. The procedural programming approach does not represent real-world models very well as any real-world entity is characterized not only by the functions it performs but also the properties (or data) it possesses. However, in procedural approach, the data and the associated functions are loosely related. Another problem with this approach is that it does not allow reusability of code. Moreover, the large programs developed using this approach is difficult to maintain, debug and extend.

To overcome the limitations of procedural programming, Object-Oriented Programming (OOP) paradigm has been developed which has revolutionized the process of software development. It includes the best features of structured programming but also some new and powerful features, including classes and objects, encapsulation, inheritance, polymorphism, abstraction, etc. These new OOP features have tremendously helped in the development of well-designed high-quality software. Examples of languages that support OOP concepts include Java, C++, C# and Visual Basic (VB). In this unit, we will discuss various concepts of OOPs, such as classes and objects, inheritance, polymorphism etc., and how these concepts are implemented in VB.NET.

3.1 OBJECTIVES

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

- Discuss the features and benefits of OOP
- Understand the concept of classes and objects
- Explain constructor and destructor
- Describe procedures, methods and event
- Explain the concept of inheritance and its implementation
- Describe the types of polymorphism, including overloading and overriding
- Understand the concept of shadowing

3.2 INTRODUCTION TO OOPS

In OOPs (Object-Oriented Programming), the programmers define not only the data, but also the operations (functions) that can be performed on it together under a single unit and thus, creating different kinds of variables known as objects. An **object** is a unit of structural and behavioral modularity that contains a set of properties (or data) as well as the associated functions. In addition, programmers can create relationships between one object and another.

As stated earlier, OOP has introduced some new and advanced features that the procedural programming lacked. The most important feature is that unlike procedural approach in which the program is divided into a number of functions, OOP divides the program into a number of objects. Some of the other features of OOP are also described here:

- OOP emphasizes on data rather than the functions or the procedures.
- OOP models the real world very well by binding the data and associated functions together under a single unit and thus, prevents the free movement of data from one function to another.
- The data of one object can be accessed by the associated functions of that object only. Other functions are not allowed to access that data. In other words, data is hidden from the outside world. However, the functions of one object can access the functions of other object.
- The objects of the entire system can interact with each other by sending messages to each other.
- The programs written in OOP are easy to maintain and extend as new objects can be easily added to the existing system whenever, required without modifying the other objects.
- OOP follows the bottom-up approach for designing the programs. That is, first objects are designed and then these objects are combined to form the entire program.

The basic idea behind OOP is shown in Figure 3.1.

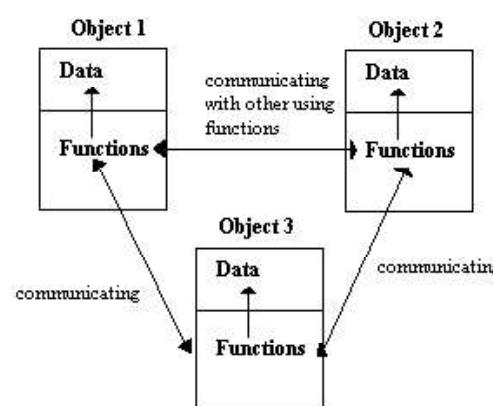


Fig. 3.1 Data and Functions in OOP

Advantages of OOP

The object-oriented programming paradigm came into use as it overcomes certain limitations of other conventional programming paradigms like the structured and unstructured paradigms. The new and advanced features of OOP, such as encapsulation, abstraction, inheritance, and polymorphism help in developing high-

NOTES

NOTES

quality software. The high-quality software can be developed due to its certain advantages. Some of the advantages of OOP are listed here.

- In OOP, writing programs with the help of objects is much similar to working with real world objects. That is, the real world objects can be conveniently represented in a program which reduces the complexity of the program and also makes the program structure clear.
- In object-oriented programs, each object is an independent and separate entity which makes modifications, locating and fixing problems in a program an easy task. In addition, any changes made inside the class do not affect the other parts of a program. Thus, object-oriented programs are easy-to-write and easy-to-maintain.
- In OOP, data integrity and data security is high as it focuses on the data, and its protection from manipulation by different parts of the program. As a result, object-oriented programs are less error-prone, more reliable and secure.
- Object-oriented programs are easy to extend as new features in a program can be added easily by introducing a few new objects without modifying the existing ones.
- OOP allows re-usability of code. That is, the objects created in one program can be re-used in other programs. In addition, new classes can be created with the help of existing ones using inheritance. It leads to faster software development and high-quality programs.
- Object-oriented programs are easier to adapt and scale, that is, large system can be created by assembling re-usable subsystems.

3.3 CLASSES AND OBJECTS

Representing the various real world objects as program elements is one of the key objectives of OOP. This objective is accomplished with the help of classes that bind data and the functions together under a single entity. A class serves as a template that provides a layout common to all of its instances known as objects. Classes and objects are the driving wheels for any programming language that implements the OOP concepts.

Objects are the small, self-contained and modular units with a well-defined boundary. An object consists of a state and behavior. The **state** of an object is one of the possible conditions that an object can exist in and is represented by its characteristics or attributes or data. The **behavior** of an object determines how an object acts or behaves and is represented by the operations that it can perform. In OOP, the attributes of an object are represented by the variables and the operations are represented by the functions.

For example, an object *biscuit* may consist of data product code *P001*, product name *Britania Biscuits*, price *20* and quantity in hand *50*. These data specify the attributes or features of the object. Similarly, consider another object *Maggi* with data product code *P002*, product name *Maggi*, price *10* and quantity in hand *20* (see Figure 3.2). In addition, the data in the object can be used by the functions, such as `check_qty()` and `display_product()`. These functions specify the actions that can be performed on data.

NOTES

A **class** is defined as a user-defined data type which contains the entire set of similar data and the functions that the objects possess. In other words, a class in OOP represents a group of similar objects. A class serves as a blueprint or template for its objects. That is, once a class has been defined, any number of objects belonging to that class can be created. The objects of a class are also known as the **instances** or the variables of that class and the process of creating objects from a class is known as **instantiation**. Note that a class does not represent an object; rather, it represents the data and functions that an object should have.

For example, a class *Product* consists of data, such as *p_code*, *p_name*, *p_price* and *qty_in_hand* which specify the attributes or features of the objects of the *Product* class. In addition, it consists of functions, such as `display_product()` and `check_qty()` that specify the actions that can be performed on data (see Figure 3.2).

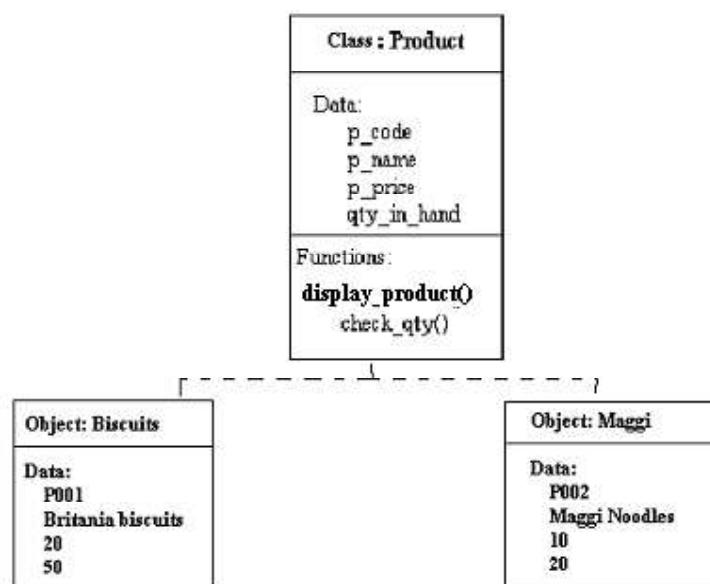


Fig. 3.2 Class and its Objects

Note: A class is only a logical abstraction that specifies what data and functions its objects will have, whereas the objects are the physical entities through which those data and functions can be used in a program.

NOTES**Encapsulation**

Encapsulation is the technique of binding or keeping together the data and the functions (that operate on them) in a single unit called a class. Encapsulation is the way to implement data abstraction. A well-encapsulated object acts as a ‘black box’ for other parts of the program. That is, it provides services to the external functions or other objects that interact with it. However, these external functions or the objects need not know its internal details. For example, in Figure 3.2, the data `p_code`, `p_name`, `p_price` and `qty_in_hand` and the functions `display_product()` and `check_qty()` are encapsulated in a class `Product`.

As stated earlier, the hidden data of a class cannot be accessed directly by the outside world. However, the member functions of the class act as a medium to access the hidden data. This process of preventing the data from the direct access by the external functions is called **data hiding**. Note that the values of the hidden data members cannot be passed to the outside world unless the functions are written to pass that information outside the class.

Creating Class

Like other user-defined data types, a class also needs to be declared and defined before using its objects in the program. A class definition specifies a new data type that can be treated as a built-in data type. The variables and functions declared within class are collectively known as **members** of the class. The variables declared in the class are known as **data members**, while the functions declared in the class are known as **member functions**.

While creating a class, the class and each of its members (variable or function) can be declared with an access specifier (or access modifier) that defines the scope to access the class or the member. The access specifier of a member controls its accessibility as well as determines the part of the program that can directly access that member of the class. VB .NET has the following access specifiers.

- **Public:** The entities declared using `Public` keyword have no restriction on their accessibility. That is, the public entities can be accessed both outside and inside the class in which it is declared. You can use the `Public` keyword while declaring any file, namespace or module.
- **Protected:** The entities declared using `Protected` keyword can be accessed only within their own class or from the derived class. You can use the `Protected` keyword only with a class.
- **Friend:** The entities declared using `Friend` keyword can be accessed only within the same program in which they have been declared or from within the assembly. Like `Public` access modifier, you can use the `Friend` keyword at the file, namespace or module level.

- **Protected Friend:** The entities declared using Protected Friend keyword have both protected and friend access. That is, they can be accessed within their own class, in the derived class or within the assembly.
- **Private:** This is the most restricted access specifier. The entities declared using Private keyword can be assessed only within their declaration context. You can use the Private keyword while declaring any file, namespace or module.

NOTES

The syntax for declaring a class in VB .NET is as follows:

```
<accessSpecifier> Class <className>
    'Define members of class here
End Class
```

Here, Class is the keyword and <className> specifies the user-defined class name.

Example 3.1: A sample class definition.

```
Public Class Book
    Private Book_title As String
    Private Book_price As Single
    Public Sub get_data(ByVal title As String, ByVal price As Single)
        'Body of method
    End Sub
    Public Sub put_data()
        'Body of method
    End Sub
End Class
```

A class must exhibit an interface so that the objects can be created from that class. The **class interface** provides a mechanism by which an application can interact with the objects of the class. It comprises properties, methods and events.

Creating Properties

Properties describe the features of the objects of a class. To store values for the properties of a class, we define member variables (or class variables) within the class. The syntax for creating properties of a class is as follows:

```
<accessSpecifier> <propertyName> As <dataType>
```

For example, the variables Book_title and Book_price (shown in **Example 3.1**) are the properties of the class Book, declared as Private.

Generally, the member variables in a class are declared using the Public, Private or Protected access specifier. If we declare the variables as Public, the variables can be accessed by all other project codes. Though it

NOTES

proves useful sometimes, it violates the principle of encapsulation which implements data hiding. Thus, the member variables in a class should preferably be declared as **Private** or **Protected** to achieve encapsulation and data hiding. Both **Protected** and **Private** variables are accessible only within the member functions of the class. In addition, the **Protected** members of a class are also accessible in the classes that are inherited from that class.

While creating objects from the class, we may need to assign values to the class variables. In case the variables are declared as **Private**, we need to use property procedures in order to pass values to the private variables in the objects and to return values from the object.

Property Procedures

A property procedure allows accessing the values of **Private** variables in an instance of the class (that is, object) with the help of **accessor methods**. There are two accessor methods: **Set** and **Get**. The **Set** accessor method is used to assign a value to the property, while the **Get** accessor method is used to retrieve the current value of the property.

The syntax for defining a property procedure is given below:

```
[Public] Property <property_name>() As <data_type>
    Get
        property_name = class_variable
        or
        Return class_variable
    End Get
    Set (ByVal Value as data_type)
        'set of statements
        class_variable = Value
    End Set
End Property
```

Here, the **class_variable** refers to the property of the class that is declared as **Private** and is to be accessed. The **value** is the keyword, which refers to the incoming value for the property. The data type of **Value** in the **Set** method must be same as the data type of return value in the corresponding **Get** method.

Note: By default, the property procedures are public, so it is optional to use the **Public** keyword while defining a property procedure. In addition, we cannot use different access specifiers for **Get** and **Set** methods.

Example 3.2: Creating a property procedure for accessing **Private** variable **Book_title** of the class **Book** defined in Example 3.1.

```
Public Property title() As String
    Get
        Return Book_title
```

End Get

OOP's Concept

```
Set (ByVal Value As String)
Book_title = Value
End Set
End Property
```

NOTES

Creating Read-Only Property

You can also create a property in a class which can only be retrieved but a new value cannot be assigned to that property. Such a property is referred to as a **read-only** property and can be created using the `ReadOnly` modifier. For such properties, we can only have `Get` accessor method in the property definition—no `Set` accessor method can be included within the definition. The syntax for creating a read-only property in a class is as follows.

```
ReadOnly Property <property_name>() As <data_type>
Get
'write code here
End Get
End Property
```

Creating Write-Only Property

In addition to read-only properties, we can also create write-only properties in a class. A **write-only** property can be assigned a value but its value cannot be retrieved. This eliminates the need of using `Get` accessor method in the property definition—only `Set` accessor method needs to be included. A write-only property can be created using the `WriteOnly` modifier in the property definition. The syntax for creating a read-only property in a class is as follows.

```
WriteOnly Property <property_name>() As <data_type>
Set (ByVal Value As data_type)
'write code here
End Set
End Property
```

Creating Methods

Methods refer to the built-in procedures of the objects of a class. A **method** is defined as a self-contained block of code that performs some function. The class methods are created by defining procedures (functions or subroutines) in the class. Like class variables, the class methods are also defined using the desired access specifier. Any method declared with `Private` keyword is accessible only within the class, while the methods declared with `Public` keyword are accessible by the objects of that class as well as the external objects. For example, two public methods `get_data()` and `put_data()` of the class `Book` (see **Example**

3.1) are defined as follows:

NOTES

```

Public Sub get_data(ByVal title As String, ByVal price As
Single)
    Book_title = title
    Book_price = price
End Sub

Public Sub put_data()
    WriteLine("Title of Book: " & Book_title)
    WriteLine("Price of Book: " & Book_price)
End Sub

```

Creating Events

Events enable objects to perform some action whenever a specific occurrence takes place. For example, suppose we have a button which when clicked twice, a particular message gets displayed in the TextBox placed beside the button. Here, Click is the event which can be handled in an appropriate event handler. Let TwiceClick be a custom event which occurs when a button is clicked twice. An event handler for this event can be coded as shown below:

```

Private Sub Button_TwiceClick(ByVal message As String) _
Handles Button.TwiceClick
    Textbox.Text = message
End Sub

```

Creating Class Objects

Once a class has been defined, we can use it to create variables of its type, that is, objects. While creating an object of a class, first we need to declare the object and then instantiate it using the New keyword. The syntax for creating object of a class is as follows:

```

Dim <object_name> As <class_name>           'Declaration
<object_name> = New <class_name>()          'Instantiation

```

Here, the <object_name> specifies the user defined object name and <class_name> is the name of the class of which object is being created. For example, we can create an object named book1 of the class Book as shown below:

```

Dim book1 As Book
book1 = New Book()

```

We can also declare and instantiate an object simultaneously using only a single statement. The syntax of declaring and instantiating an object in a single statement is as follows:

```
Dim <object_name> As <class_name> = New <class_name>()
```

OOP's Concept

For example, the object book1 can be declared and instantiated at the same time using the following statement:

```
Dim book1 As Book = New Book()
```

A coding shortcut for the above statement is given as follows:

```
Dim book1 As New Book()
```

NOTES

Accessing Class Members

As we know that the members of a class can be directly accessed inside the class using their names. However, accessing a member outside the class depends on its access specifier. The public members of a class can be accessed outside the class directly using the object name and dot operator ‘.’. The dot operator associates a member with the specific object of the class. The syntax for accessing a Public class variable outside the class is as follows:

```
<object_name>.class_variable
```

The syntax for accessing a Public class method outside the class is as follows:

```
<object_name>. <method_name>([parameter_list])
```

For example, the object book1 of class Book can call the public methods get_data() and put_data() of class as shown below:

```
book1.get_data()  
book1.put_data()
```

As told earlier, the private members of class can be accessed outside the class through property procedures. In order to assign a value to the private variable of an object, we need to call the Set method of the corresponding property procedure using the syntax shown below:

```
<object_name>. <property_name> = {<variable> | <value>}
```

For example, if we want to set the price of book1 object as 600, we can do so using the following statement:

```
book1.price = 600
```

Similarly, we can retrieve the value of a private class variable outside the class by invoking the Get method of the corresponding property procedure, using the following syntax:

```
<variable> = <object_name>. <property_name>
```

For example, if we want to store the price of book1 object in a variable Rate, we can do so using the following statement:

```
Rate = book1.price
```

A structure is a derived data type which consists of collection of different data types such as int, char, float and even structure also. Consider an example for keeping information about student, we have to store his/her name, registration number, age, address, class etc. Similarly for keeping

NOTES

information about employee we may have his/her employee code, date of joining, salary etc. All these attributes related to student or employee cannot be stored in array alone. Even if we wish to store these attribute using array we will have to use separate array for each attribute say one for salary, one for employee code and so on. Again managing this array separately won't be an easy task. In such situations we can use structures. We understand this new concept of C programming using a program.

The general syntax of creating a structure is:

```
struct tagname
{
    member variable1;
    member variable2;
    member variable3;
    ....
    member variablen;
};
```

The struct is a keyword that is used to create a structure and tagname is the name of the structure that conforms to the rule of writing identifier. For understand number of points about the structure we write a small program and then tell you various features of structure.

```
struct student
{
    char name[10];
    int rollno;
};

main()
{
    struct student t;
    clrscr();
    printf("Enter name for student\n");
    scanf("%s",t.name);
    printf("Enter rollno for student\n");
    scanf("%d",&t.rollno);
    printf("Name=%s\n",t.name);
    printf("Age=%d\n",t.rollno);
    getch();
}
OUTPUT:
```

```
Enter name for student  
Ramchander  
Enter rollno for student  
40  
Name=Ramchander  
rollno=40
```

OOP's Concept

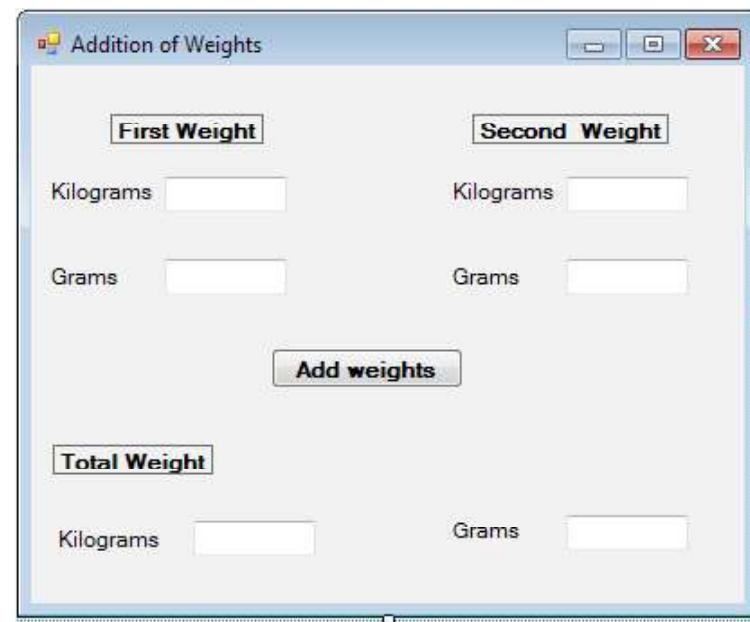
NOTES

1. Structure is a heterogeneous data structure. Remember array is a homogenous data structure i.e. all the elements of an array must be of the same type. You cannot mix int with float or float with char. This is not the case with structure ie structure can have these elements of any data type even they may have other user defined data types as their elements.
2. Structure is defined by struct keyword. Following struct any valid identifier which conforms to identifier writing rules is allowed. In this program we have declared a structure named person. We could have chosen any other name also say student or employee.
3. The structure is opened using { and closed with } followed by a semicolon (;).
4. The structure declared above contains two member elements: first one is the character array name of length 10 and second one is rollno of type int. As we know
int takes two bytes in memory (for DOS) so total size for this structure is 12 bytes.
5. The declaration alone does not reserve memory space. It simply tells the compiler about the prototype of the structure. The actual definition i.e. creation of a variable of structure data type allocate memory.

Creating a Sample Application using Classes and Objects

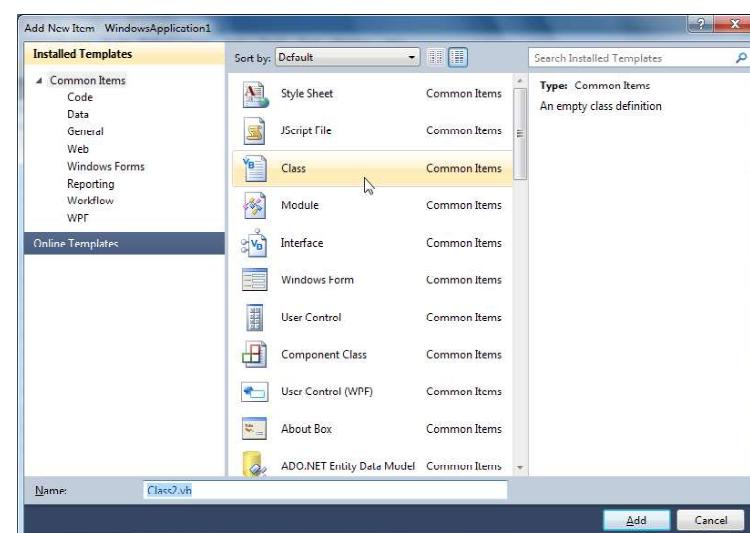
The concept of classes and objects can be used in any application, whether it is a Console application, Windows Forms application or any other application. In this section, we discuss the stepwise procedure to design a Windows Forms application employing the concept of classes and objects.

Suppose we have to design a form as shown in Figure 3.3. In this form, a user enters the values for kilograms and grams for the two weights in the respective textboxes under the **First Weight** and **Second Weight** label headings. When user clicks the **Add weights** button, the weights entered by the user are summed up and the resulting kilograms and grams values are displayed in the respective textboxes under **Total Weight** label heading.

NOTES**Fig. 3.3 A Sample Form**

To design this Windows Forms application implementing the concept of classes and objects, follow these steps.

1. Open a new Windows Forms application.
2. Design the form like as shown in Figure 3.3 by placing the required controls on it and setting their appropriate properties.
3. To begin creating a new class, click the **Project** menu and then click **Add Class**. Alternatively, right-click on the name of application in the Solution Explorer window, point to **Add** and then click **Class** from the submenu that appears. This displays the Add New Item dialog box (see Figure 3.4).

**Fig. 3.4 Add New Item Dialog Box**

4. Click **Class** option (by default, highlighted) from the middle pane of dialog box.
5. Type the name for class module in the **Name TextBox**. For example, we have used the name **weight** for our class.
6. Click **Add**. The class gets added in the application and the Code Editor window appears. Here, you can define your class.
7. Declare the two private properties (**kg** and **gm**) of class **weight** in the line after the **Public Class weight** statement, as shown in Figure 3.5.
8. Add two property procedures, **kilo()** and **gram()** for private properties **kg** and **gm**, respectively (see Figure 3.3) so that these properties can be accessed outside the class module.
9. Now, add a public method **add_weight()** in the class that computes the sum of two weight objects and returns the resultant weight object. Note that as the method is supposed to return something, we have to use function not the sub procedure to implement the class method. Figure 3.5 shows the Code Editor window with the complete code for the class **weight**.

NOTES

```

Form1.vb    weight.vb*  Form1.vb [Design]
weight
Public Class weight
    Private kg, gm As Integer
    Public Property kilo() As Integer
        Get
            Return kg
        End Get
        Set(ByVal value As Integer)
            kg = value
        End Set
    End Property
    Public Property gram() As Integer
        Get
            Return gm
        End Get
        Set(ByVal value As Integer)
            gm = value
        End Set
    End Property
    Public Function add_weight(ByVal w2 As weight) As weight
        Dim w = New weight()
        w.gm = gm + w2.gm
        w.kg = w.gm \ 1000      'integer division
        w.gm = w.gm Mod 1000
        w.kg += kg + w2.kg
        Return w
    End Function
End Class

```

Fig. 3.5 Weight Class Definition

10. Switch to Form design window by clicking on the name of form in the Solution Explorer Window.
11. In the **Click** event of the *Add weights* button, write the code shown in Figure 3.6.

NOTES

```

(General) (Declarations)
Public Class Form1
    Private Sub but_add_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles but_add.Click
        Dim w1 = New weight()      'creating object of class weight
        Dim w2 = New weight()
        Dim w3 = New weight()

        'store the values entered in the textboxes under First Weight label heading
        'into private variables of object w1
        w1.kilo = Val(txt_kg1.Text)      'invoking Set method of kilo property procedure
        w1.gram = Val(txt_gm1.Text)      'invoking Set method of gram property procedure

        'store the values entered in the textboxes under Second Weight label heading
        'into private variables of object w2
        w2.kilo = Val(txt_kg2.Text)      'invoking Set method of kilo property procedure
        w2.gram = Val(txt_gm2.Text)      'invoking Set method of kilo property procedure

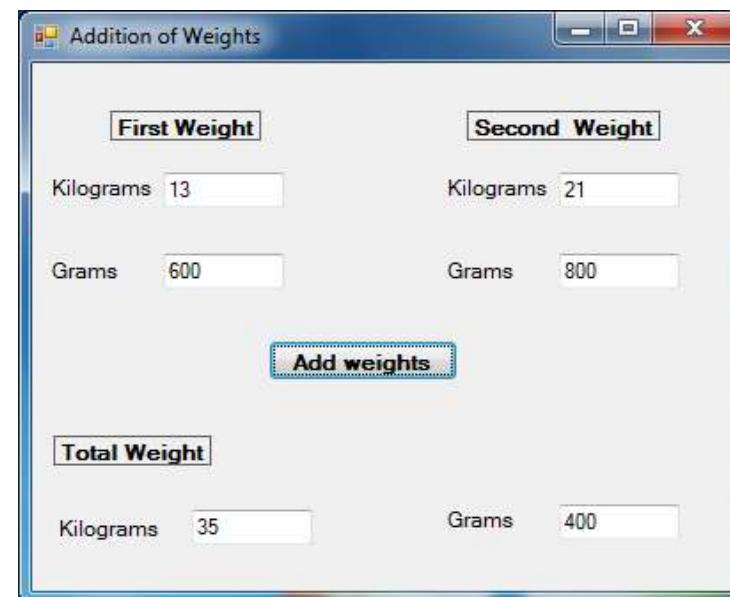
        w3 = w1.add_weight(w2)          'invoking the add_weight() function of class

        'retrieve the values of private variables of object w3 and display them in the
        'textboxes under the Total Weight label heading
        txt_kg.Text = w3.kilo          'invoking Get method of kilo property procedure
        txt_gm.Text = w3.gram          'invoking Get method of gram property procedure
    End Sub
End Class

```

Fig. 3.6 Code Editor Window of Form Control

12. Press **F5** key or click the **Start Debugging** (▶) button on the Standard toolbar to run the application. The form appears on the screen.
13. Enter the desired data in the textboxes under *First Weight* and *Second Weight* label headings and then test the application by clicking on *Add weights* button. Figure 3.7 shows the form after running the application.

**Fig. 3.7** Output after Running the Application

Note that you can also define your class within the module in case of Console application and on the `Load` or any other event of form in case of Windows Forms application. This implies that you do not have to add a separate class module in your application, as we have described in step 3. Defining a class within the module or in one of the form events is useful in case your application contains

only a single form/module or if class members are not required to be accessed by other forms/modules. For simplicity, from now onwards, we will use the Console application to illustrate various OOP concepts and define the class within the module itself.

NOTES

3.4 CONSTRUCTOR AND DESTRUCTOR

In VB 2010, there are two important members in a class: Constructor and Destructor, and Shared Members. In this section, we discuss the constructor and destructor members. Shared members of a class are discussed in *Section 4.9*.

Constructor

A constructor is a special method of a class that is automatically invoked whenever, an instance of the class is created. It is used to initialize the objects of the related class at the time of their creation and set the parameters. An important thing to note about a constructor is that it never returns a value. The reason behind this is that a constructor is invoked automatically by the system and hence, no program has been defined for it to return anything. Note that a constructor must be declared as public (or protected) so that it can be accessed while creating the objects.

A constructor may be written without containing any parameter list, referred to as **default (or empty) constructor**. A default constructor initializes all objects of the related class with the same values. In addition, a constructor can also accept one or more parameters to initialize the objects with those parameters. Such a constructor is referred to as **parameterized constructor** and is used to initialize the objects with different values.

Note: VB automatically creates an empty constructor for each class. Thus, if you require only an empty constructor in your class, you need not define it explicitly. This implicit constructor initializes the objects of a class with garbage values.

In VB .NET, the constructor method is defined as a subroutine named `New()`. As soon as the class is instantiated, the subroutine `New()` is called. The syntax of defining a constructor within a class is as follows.

```
Public Sub New([<List of arguments>])
    'write code here
End Sub
```

We can use multiple constructors with different number, data type or order of parameters in a single class and it is known as **constructor overloading**. For example, a class can have both default and parameterized constructor in its definition. Constructor overloading enables multiple constructors to initialize different objects of a class differently.

Example 3.3: A program to demonstrate the use of constructor in a class.

```
Module Module1
    Public Class weight
```

NOTES

```

Private kg, gm As Integer
    Public Sub New()           'Default constructor
        kg = 0
        gm = 0
    End Sub

    Public Sub New(ByVal kilo As Integer, ByVal gram As
Integer)
        kg = kilo      'parameterized constructor
        gm = gram
    End Sub

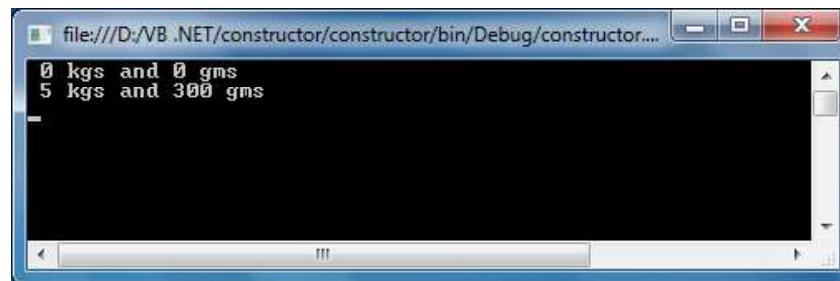
    Public Sub show()
        Console.WriteLine(" " & kg & " kgs and " & gm & "
gms")
    End Sub

End Class

Sub Main()
    Dim w1 = New weight() 'calling default constructor
    Dim w2 = New weight(5, 300) 'calling parameterized
constructor
    w1.show()
    w2.show()
    Console.Read()
End Sub

End Module

```

The output of the program is**Destructor**

When objects are created, they occupy certain memory. In addition, an object may need to use other resources like files during its life time. These resources and the memory allocated to objects must be released when an object is destroyed (or goes out of scope). This is accomplished by another special class method called **destructor** that is automatically invoked to release all the resources and memory acquired by the object when the object is finally destroyed.

Note: A destructor is the last method run by any class.

A VB .NET program can include two types of destructors in a class, including `Finalize` and `Dispose`. The `Finalize` destructor is more frequently used as compared to `Dispose` destructor.

OOP's Concept

Using the `Finalize` Destructor

The `Finalize` method is invoked automatically by the .NET framework when the system determines that the object is no more required. This implies that one need not explicitly call the `Finalize` method by itself. The syntax for using `Finalize` destructor in a class is as follows.

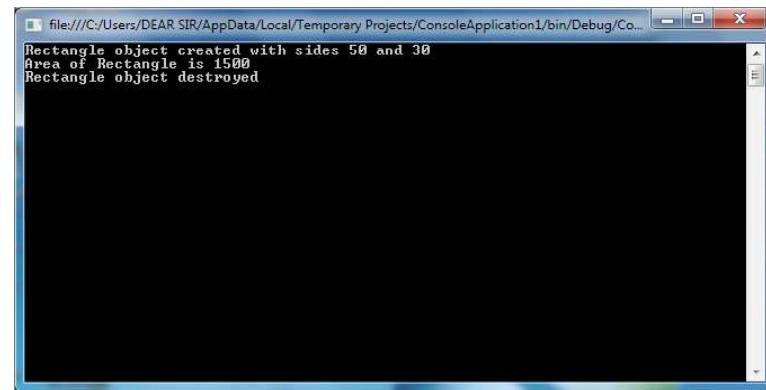
```
Protected Overrides Sub Finalize()  
    'write code here  
End Sub
```

Example 3.4: A program to demonstrate the use of `Finalize` destructor.

```
Module Module1  
    Public Class Rectangle  
        Private length, breadth As Integer  
        'Defining constructor  
        Public Sub New(ByVal l As Integer, ByVal b As Integer)  
            length = l  
            breadth = b  
            Console.WriteLine("Rectangle object created with sides  
{0} and {1}", l, b)  
        End Sub  
        'Defining Finalize destructor  
        Protected Overrides Sub Finalize()  
            Console.WriteLine("Rectangle object destroyed")  
            Console.Read()  
        End Sub  
        Public Sub Compute_Area()  
            Console.WriteLine("Area of Rectangle is {0}", length  
* breadth)  
        End Sub  
    End Class  
  
    Sub Main()  
        Dim rec As New Rectangle(50, 30)  
        rec.Compute_Area()  
    End Sub  
End Module
```

NOTES

The output of the program is



Using the **Dispose** destructor

The **Dispose** method is also used to clean up for the object when it is no longer needed. However, unlike **Finalize** method, it needs to be invoked manually for each object whenever you want. The **Dispose** method is declared in the interface **IDisposable** defined by the .NET framework class libraries. Thus, to use this method, one needs to implement the **IDisposable** interface in the class and then define the **Dispose** method of interface, which contains the code for cleaning-up for the objects. The syntax for using **Dispose** method is as follows.

```
<accessSpecifier> Sub <method_name>() Implements
IDisposable.Dispose
    'write code here
End Sub
```

Here, we rewrite the same program as in **Example 3.4** but using **Dispose** method instead of **Finalize**.

Example 3.5: A program to demonstrate the use of **Dispose** destructor.

```
Module Module1
    Public Class Rectangle      'class needs to implement
        the interface
        Implements IDisposable      'whose method is to be used
        in class
        'define the private variables and constructor, as in
        Example 3.4
        .
        .
        .
        'defining dispose destructor
        Public Sub Dispose() Implements IDisposable.Dispose
```

```

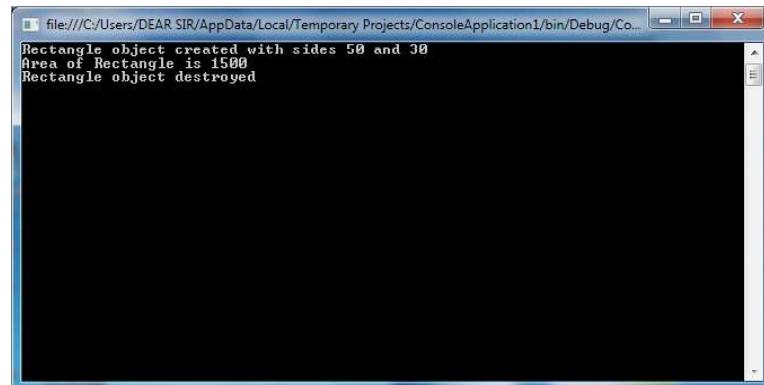
Console.WriteLine("Rectangle object destroyed")
Console.Read()
End Sub
    'define Compute_Area(), as in Example 3.4
    .
    .
    .
End Class
Sub Main()
Dim rec As New Rectangle(50, 30)
rec.Compute_Area()
rec.dispose()      'explicitly invoking destructor
End Sub
End Module

```

OOP's Concept

NOTES

The output of the program is



3.5 INHERITANCE

Inheritance is a mechanism of deriving a new class (or derived class) from the old class (or base class) in such a way that the new class inherits all the members (variables and methods) of the old class. In other words, inheritance facilitates a class to acquire the properties and functionality of another class. The new class depicts the acquired properties and behavior of the existing class as well as its own unique properties and behavior. Inheritance allows code reusability, that is, it facilitates classes to reuse the existing code. The new class acquires the members of old class that are already tested and debugged. Hence, inheritance saves time as well as increases the reliability.

Base Classes and Derived Classes

In inheritance, the class that is inherited by the new class is called **base class** or **super class** or **parent class**. The class that inherits the data members of the old

NOTES

class is called **derived class** or **sub class** or **child class**. For example, Figure 3.8 shows four classes, named, animal, carnivore, herbivore and omnivore. The class animal is a base class inherited by the derived classes carnivore, herbivore and omnivore. The derived classes carnivore, herbivore and omnivore inherit all the properties as well as functionality of the base class animal.

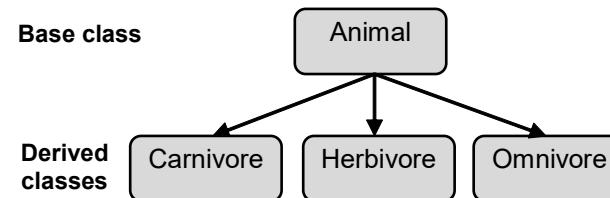


Fig. 3.8 Inheritance

Note: The constructors of a base class are not inherited in its derived classes.

The general form of inheriting a derived class from base class is as follows.

```

<accessSpecifier> Class <baseClass>
    'define body of base class
End Class

<accessSpecifier> Class <derivedClass>
    Inherits <baseClass>
    'define body of derived class
End Class
  
```

Here, the `Inherits` statement in the definition of derived class specifies that the `<derivedClass>` is based on the `<baseClass>`.

The derived class inherits all the public and protected data members and methods of the base class. In addition, the derived class can also contain its own members. However, an exception to this is the constructors of the base class, which are not inherited in its derived class.

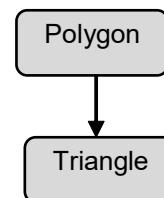
Note: The derived classes inherit and can extend the methods, properties, fields and events of the base class.

3.5.1 Types of Inheritance

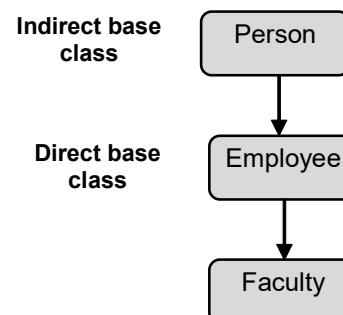
Depending on the number of classes involved and the way the classes are inherited, inheritance is of four types, which are described as follows.

- **Single Inheritance:** It is the most basic type of inheritance in which only one derived class inherits from a single base class. Figure 3.9 illustrates single inheritance in which the class `Triangle` inherits from the base class `Polygon`.

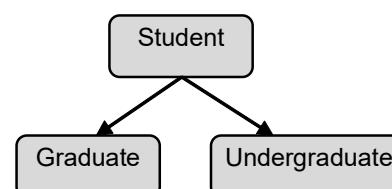
NOTES

*Fig. 3.9 Single Inheritance*

- **Multilevel Inheritance:** When one class is inherited from another class, which in turn is inherited from some other class, it is referred to as multilevel inheritance. Figure 3.10 illustrates multilevel inheritance in which the class Faculty inherits from the class Employee which inherits from the class Person. This implies that Person is the indirect base class of Faculty.

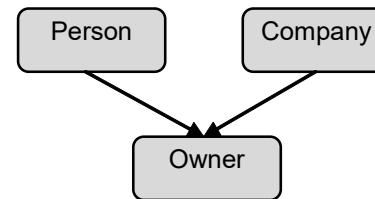
*Fig. 3.10 Multilevel Inheritance*

- **Hierarchical Inheritance:** It is a type of inheritance in which two or more classes are inherited from the same base class. In hierarchical inheritance, the members of the base class are common to all of its derived classes. Figure 3.11 illustrates hierarchical inheritance in which two classes Graduate and Undergraduate are derived from the same base class Student.

*Fig. 3.11 Hierarchical Inheritance*

- **Multiple Inheritance:** When a derived class inherits from more than one base class simultaneously, it is referred to as multiple inheritance. In multiple inheritances, the derived class inherits the members of all of its base classes. Figure 3.12 illustrates multiple inheritances in which the derived class Owner is inherited from two base classes, Person and Company.

NOTES

**Fig. 3.12** Multiple Inheritance

Note: VB .NET does not directly support multiple inheritance because of the involvement of multiple base classes. It can indirectly implement multiple inheritance through interfaces.

3.5.2 Implementation of Basic Inheritance

In this section, we provide an example program to illustrate how to inherit a derived class from a base class and how to access the data members and methods of base class using the derived class object.

Example 3.6: A program to demonstrate the implementation of inheritance.

```

Module Module1
    Public Class person           'base class
        Protected name As String
        Protected age As Integer
        Public Sub get_data(ByVal str As String, ByVal ag As Integer)
            name = str
            age = ag
        End Sub
        Public Sub display()
            Console.WriteLine("Invoking display method of base class")
            Console.WriteLine(" Name: " & name)
            Console.WriteLine(" Age: " & age)
        End Sub
        End Class

        Public Class employee 'derived class
    Inherits person
        Private emp_no As String      'new member of derived class
        'two new methods of derived class
        Public Sub input_data(ByVal eno As String)
            emp_no = eno
        End Sub
        Public Sub show()
            Console.WriteLine()
        End Sub
    End Class
  
```

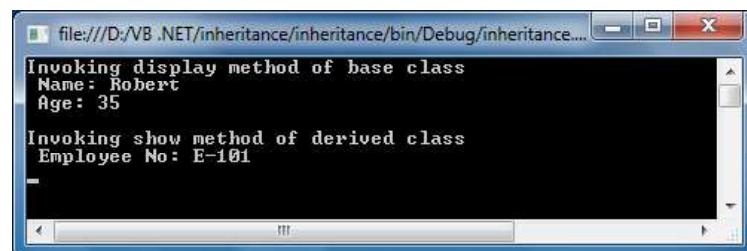
```
Console.WriteLine("Invoking show method of derived  
class")  
Console.WriteLine(" Employee No: " & emp_no)  
End Sub  
End Class
```

OOP's Concept

```
Sub Main()  
    Dim emp = New employee()      'creating derived class  
object  
    emp.get_data("Robert", 35)   'invoking base class methods  
    emp.display()                'using derived class  
object  
  
    emp.input_data("E-101")     'invoking derived class  
methods  
    emp.show()                  'using derived class object  
    Console.ReadLine()  
End Sub  
End Module
```

NOTES

The output of the program is



Using Constructor in Inheritance

As stated earlier, the constructor of base class is not inherited in the derived class. The derived class defines its own constructor in order to initialize its new members (if any). However, to initialize the members inherited from the base class, it needs to invoke the base class constructor.

The base class constructor can be called using the following statement:

```
MyBase.New(<parameter_list>)
```

Notice that this should be the first statement in the body of derived class constructor. In this statement, `MyBase` is the keyword and the `parameter_list` includes the arguments corresponding to inherited members of base class (as used in base class constructor) as well as the parameters of derived class (if any).

Note: If base class contains only an empty constructor, there is no need to invoke it inside the derived class constructor.

NOTES

To understand the use of constructors in inheritance, consider **Example 3.7** which is the modified form of the program given in **Example 3.6**. In this example, we have replaced the `get_data()` and `input_data` method of base and derived class, respectively, with constructors.

Example 3.7: A program to demonstrate the use of constructors in inheritance.

```

Module Module1
    Public Class person
        Protected name As String
        Protected age As Integer
        'Parameterized constructor of base class
        Public Sub New(ByVal str As String, ByVal ag As Integer)
            name = str
            age = ag
        End Sub
        'define display() method same as in Example 3.6
    End Class
    Public Class employee
        Inherits person
        Private emp_no As String      'new member of derived
        'constructor of derived class
        Public Sub New(ByVal str As String, ByVal ag As Integer,
ByVal eno As String)
            MyBase.New(str, ag)      'invoking the base
            class constructor
            emp_no = eno           'initializing derived
            class member
        End Sub
        'define show() method same as in Example 3.6
    End Class

    Sub Main()
        'creating derived class object
        Dim emp = New employee("Robert", 35, "E-101")
        emp.display()
        emp.show()
        Console.ReadLine()
    End Sub
End Module

```

This program produces the same output as the program given in Example 3.6.

In VB 2010, each class can serve as base class, by default. However sometimes, it is desirable to design a class that cannot be inherited by another class. To prevent inheritance, a class must explicitly declare that it cannot be inherited. This can be achieved by using the inheritance modifier `NotInheritable` within the class definition. The main objective of a `NotInheritable` class is to provide some implementation that can be used within the class only.

The syntax for defining a `NotInheritable` class is as follows:

```
NotInheritable Class <class_name>
    'define body of class here
End Class
```

Here, the class `<class_name>` is declared as `NotInheritable`. If an attempt is made to derive a class from `<class_name>`, an error will arise.

Note: In addition to `NotInheritable`, there is one more inheritance modifier `MustInherit` that should be used with a class which is intended to be used as a base class only.

NOTES

3.6 POLYMORPHISM

Polymorphism (a Greek word meaning *having multiple forms*) is one of the important features of OOP that allows an entity (method or property) to be represented in various forms. Using polymorphism, we can use a procedure in many different ways depending on the requirements. For example, we can define a procedure to calculate the area of a geometrical shape and use it with different parameters for calculating area of different shapes, like square, rectangle or circle.

Polymorphism allows a class to have multiple functions with same name but with different number, order or type of arguments. It has been divided into two categories: Compile-time polymorphism/overloading and Run-time polymorphism/overriding.

3.6.1 Overloading

Compile-time polymorphism (also called **early binding**) is a process in which a compiler recognizes the method arguments during compilation of the program and accordingly, binds the appropriate method with the object at the compile time. The compile-time polymorphism can be implemented using overloaded methods and operators. In this section, we discuss method overloading and operator overloading.

Method Overloading

Method overloading is a way to implement compile-time polymorphism that allows multiple methods to share the same name but with different arguments. When an

NOTES

object invokes an overloaded method, the compiler knows by itself which method to bind with based on the signature used in method call. That is, the compiler identifies the method either on the basis of the number of arguments, the data type of the arguments or the order of the data type of the arguments passed to the method.

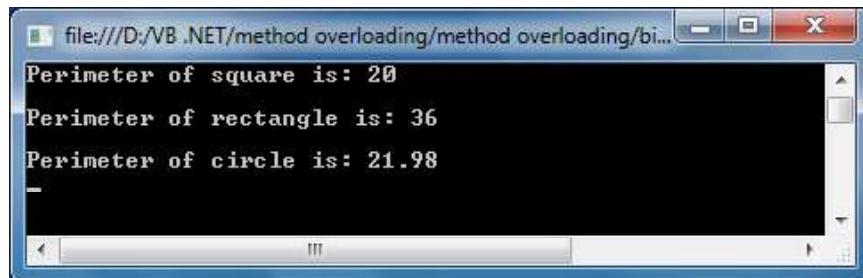
Example 3.8: A program to demonstrate the concept of method overloading.

```

Module Module1
    Class perimeter
        'overloaded method peri()
        Public Function peri(ByVal side As Integer)
            Dim per As Single = 4 * side      'computing perimeter
            of a square
            Return per
        End Function
        'computing perimeter of a rectangle
        Public Function peri(ByVal l As Single, ByVal b As
        Single) As Single
            Dim per As Single = 2 * (l + b)
            Return per
        End Function
        'computing perimeter of a circle
        Public Function peri(ByVal rad As Double) As Double
            Dim per As Double = 2 * 3.14 * rad
            Return per
        End Function
        End Class

        Sub Main()
            Dim shape = New perimeter()
            Console.WriteLine("Perimeter of square is: {0}",
            shape.peri(5))
            Console.WriteLine()
            Console.WriteLine("Perimeter of rectangle is: {0}",
            shape.peri(10.5, 7.5))
            Console.WriteLine()
            Console.WriteLine("Perimeter of circle is: {0}",
            shape.peri(3.5))
            Console.Read()
        End Sub
    End Module

```

**NOTES**

Operator Overloading

One of the important objectives of OOP is that the variables of user-defined data types like objects of a class can be treated like the variables of built-in data types. That is, all the arithmetic and logical operations can be performed on the objects of a class in the same way as performed on simple variables. To perform operations on simple variables, some built-in operators, such as '+', '-', '*', '/', '<', '>', '=' etc., are provided with each operator having a specific functionality. To use these operators with the user-defined data types, VB .NET provides a way by which the functionality of an operator can be changed according to user-defined data types and additional meaning can be given to the operator. This is what is known as **operator overloading**.

Though most operators in VB .NET can be overloaded, it is not possible to overload all the operators. Few operators in VB .NET that can be overloaded are listed in Table 3.1.

Table 3.1 Operators that can be Overloaded

Type	Operator
Unary	Not (logical negation), + (unary plus), - (unary minus), IsTrue, IsFalse
Binary	arithmetic operators (+, -, *, /, Mod, \, ^), << (arithmetic shift left), >> (arithmetic shift right), & (string concatenation)
Comparison	< (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), <> (not equal to), = (equal to)
Type conversion	CType

Some operators that cannot be overloaded are listed here.

- Logical comparison operators, including Or, And, Xor, OrElse and AndAlso
- Assignment operators, including +=, -=, /=, *=, <<=, >>=, ^= and &=
- Other operators, including New, Like, TypeOf, GetType, Is, IsNot, AddressOf, dot(.), and ?:.

NOTES

An operator overload method is declared in the same manner as any other class method except the **Operator** keyword is used instead of **Function** or **Sub** in the method declaration. Moreover, the overloaded operators are always shared methods (indicated by the **Shared** keyword preceding the **Operator** keyword in the method declaration), which must return a value. This return value needs not necessarily be of the same type as the types on which operation is performed. The general form of an operator overload method is as follows:

```
<accessSpecifier> Shared Operator <oprtr> (<arguments>)
As <returnType>
'define body of method here
End Operator
```

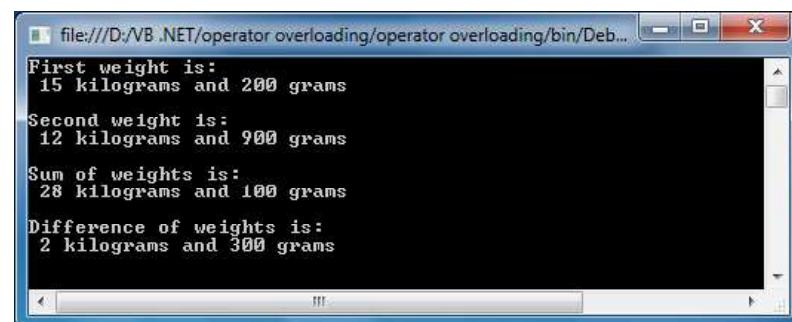
Example 3.9: A program to demonstrate the concept of operator overloading.

```
Module Module1
    Public Class weight
        Private kg, gm As Integer
        Public Sub New()
            kg = 0
            gm = 0
        End Sub
        Public Sub New(ByVal k As Integer, ByVal g As Integer)
            kg = k
            gm = g
        End Sub
        Public Sub show()
            Console.WriteLine(" {0} kilograms and {1} grams", kg,
gm)
        End Sub
        Public Shared Operator +(ByVal w1 As weight, ByVal w2 As
weight) As weight      'overloading + operator
            Dim temp = New weight()
            temp.gm = w1.gm + w2.gm
            temp.kg = temp.gm \ 1000
            temp.gm = temp.gm Mod 1000
            temp.kg += w1.kg + w2.kg
            Return temp
        End Operator
        Public Shared Operator -(ByVal w1 As weight, ByVal w2 As
weight) As weight      'overloading - operator
            Dim temp = New weight()
            If (w1.gm < w2.gm) Then
```

```
w1.gm += 1000
w1.kg -= 1
End If
temp.gm = w1.gm - w2.gm
temp.kg = w1.kg - w2.kg
Return temp
End Operator
End Class

Sub Main()
Dim w1 = New weight(15, 200)
Dim w2 = New weight(12, 900)
Dim w3, w4 As New weight()
Console.WriteLine("First weight is:")
w1.show()
Console.WriteLine()
Console.WriteLine("Second weight is:")
w2.show()
Console.WriteLine()
w3 = w1 + w2      'calling overloaded operator +
Console.WriteLine("Sum of weights is:")
w3.show()
Console.WriteLine()
w4 = w1 - w2      'calling overloaded operator -
Console.WriteLine("Difference of weights is:")
w4.show()
Console.Read()
End Sub
End Module
```

The output of the program is



NOTES

NOTES**3.6.2 Overriding Methods and Properties**

A base class method (or property) can be inherited in the derived class; however, sometimes it may need to be redefined according to the requirement in the derived class. An important feature of OOP that allows redefining a base class method in the derived class is **overriding**. This implies that the functionality of a method declared inside the base class can be overridden in the derived classes. The entire function of the base class can be replaced by a set of new implementation in the derived class.

Overriding is also referred to as **run-time polymorphism** (or **late binding**), as the compiler recognizes which specific implementation of method to invoke during the execution of program and thus, binds the appropriate method with the object at the run-time itself. To implement overriding, the keyword `Overridable` is used within the declaration of base class method (or property) and the keyword `Overrides` is used within the implementation of derived class method (or property). The general form of overriding a method of base class in derived class is as follows:

```
'base class definition
<accessSpecifier> Class <baseClass>
    <accessSpecifier> Overridable Sub <methodName>()
        'define body of method here
    End Sub
End Class
.
.
.

'derived class definition
<accessSpecifier> Class <derivedClass>
    Inherits <baseClass>
    <accessSpecifier> Overrides Sub <methodName>()
        'define body of method here
    End Sub
End Class
```

Notice that the base class method (that is overridden) and its redefined versions in the derived classes must have the same prototype, that is, the name of method, number and data type of arguments as well as return type (if any) must be same. However, if the prototype differs, the compiler considers these functions as overloaded functions and the overriding mechanism is ignored.

Once a base class method has been redefined in the derived class, any call to that method by the derived class object automatically invokes the implementation of that method defined in the derived class, instead of base class implementation. However, if a method declared with keyword `Overridable` in the base class

is not redefined in the derived class, a call to that method uses the base class implementation of that method. Moreover, the **Overridable** method of base class can also be accessed directly in the derived class using the following statement:

MyBase.<method_name>

Example 3.10: A program to demonstrate the concept of overriding.

NOTES

```

Module Module1

    Public Class division
        Protected num, den As Single
        Public Sub New(ByVal n As Single, ByVal d As Single)
            num = n
            den = d
        End Sub
        Public Overridable Sub divide()
            Dim result As Single = num / den
            Console.WriteLine("divide() of base class")
            Console.WriteLine("Result of division operation {0}/{1} = {2}", num, den, result)
        End Sub
    End Class

    Public Class div      'derived class
        Inherits division
        Public Sub New(ByVal n As Single, ByVal d As Single)
            MyBase.New(n, d)  'invoking constructor of base class
        End Sub
        Public Overrides Sub divide()   'overriding divide()
            If (den <> 0) Then
                MyBase.divide()      'invoking divide() of base class
            Else
                Console.WriteLine("Division operation cannot be performed")
            End If
        End Sub
    End Class
    Sub Main()
        Dim d1 = New division(8.5, 2)      'base class object
        Dim d2 = New div(4.5, 0)           'derived class object
        Dim d3 = New div(6.9, 3)
        d1.divide() 'divide() of base class called
    End Sub

```

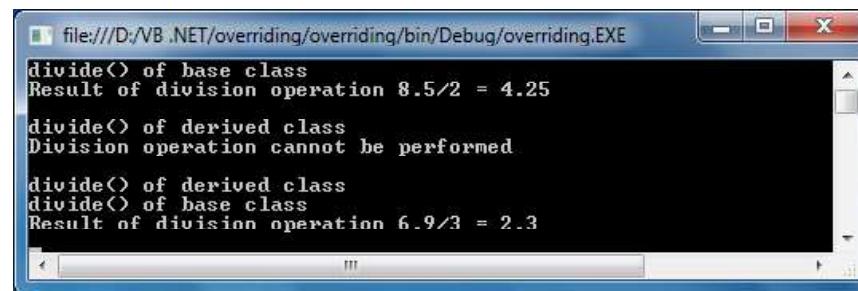
NOTES

```

        Console.WriteLine()
d2.divide()      'overriding: divide() of derived
class called
        Console.WriteLine()
d3.divide()      'overriding: divide() of derived
class called
        Console.Read()
End Sub
End Module

```

The output of the program is



3.7 SHADOWING

Shadowing is used to provide a new implementation to base class member without overriding it, which means that original implementation of base class member gets shadowed (hidden) with the new implementation of base class member provided in derived class.

```

class Parent
{
    // Declaring instance variable with name 'abc'
    String abc = "Parent's Instance Variable";
    public void printInstanceVariable()
    {
        System.out.println(abc);
    }
    public void printLocalVariable()
    {
        // Shadowing instance variable 'abc' with a local
        // variable with the same name
        String abc = "Local Variable";
        System.out.println(abc);
        // If we still want to access the instance variable,
        we do that by using 'this.abc'
    }
}

```

```
        System.out.println(this.abc);  
    }  
}
```

OOP's Concept

In the above example, there is an instance variable named abc, and inside method printLocalVariable(), we are shadowing it with the local variable abc.

NOTES

Check Your Progress

1. What are the various advanced features of OOP that help in developing high-quality software?
2. Define the term method.
3. What is the purpose of creating events in VB.NET?
4. What is the significance of polymorphism?

3.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The various advanced features of OOP that help in developing high-quality software are encapsulation, abstraction, inheritance, and polymorphism.
2. A method is defined as a self-contained block of code that performs some function. The class methods are created by defining procedures (can be functions or subroutines) in the class.
3. Events enable objects to perform some action whenever a specific occurrence takes place. For example, suppose we have a button which when clicked twice, a particular message gets displayed in the textbox placed beside the button. Here, Click is the event which can be handled in an appropriate event handler.
4. Polymorphism allows a class to have multiple functions with same name but with different number, order or type of arguments.

3.9 SUMMARY

- OOPS (Object-Oriented Programming), the programmers define not only the data, but also the operations (functions) that can be performed on it together under a single unit and thus, creating different kinds of variables known as objects.
- The object-oriented programming paradigm came into use as it overcomes certain limitations of other conventional programming paradigms like the structured and unstructured paradigms.
- Classes and objects are the driving wheels for any programming language that implements the OOP concepts.

NOTES

- Objects are the small, self-contained and modular units with a well-defined boundary that consists of a state and behavior.
- A class is defined as a user-defined data type which contains the entire set of similar data and the functions that the objects possess.
- Encapsulation is the technique of binding or keeping together the data and the functions (that operate on them) in a single unit called a class.
- The concept of classes and objects can be used in any application, whether it is a Console application, Windows Forms application or any other application.
- In VB (Visual Basic) 2010, there are two important members in a class: Constructor and Destructor, and Shared Members.
- A constructor is a special method of a class that is automatically invoked whenever an instance of the class is created. A destructor is another special class method that is automatically invoked to release all the resources and memory acquired by the object when the object is finally destroyed.
- Inheritance is a mechanism of deriving a new class (or derived class) from the old class (or base class) in such a way that the new class inherits all the members (variables and methods) of the old class.
- The class that is inherited by the new class is called base class or super class or parent class, and the class that inherits the data members of the old class is called derived class or sub class or child class.
- Depending on the number of classes involved and the way the classes are inherited, inheritance is of four types: Single, Multilevel, Hierarchical and Multiple.
- Polymorphism (a Greek word meaning having multiple forms) is one of the important features of OOP that allows an entity (method or property) to be represented in various forms. Using polymorphism, we can use a procedure in many different ways depending on the requirements.
- It has been divided into two categories: Compile-time polymorphism/overloading and Run-time polymorphism/overriding.
- Compile-time polymorphism (also called early binding) is a process in which a compiler recognizes the method arguments during compilation of the program and accordingly, binds the appropriate method with the object at the compile time.
- Method overloading is a way to implement compile-time polymorphism that allows multiple methods to share the same name but with different arguments.
- A base class method (or property) can be inherited in the derived class; however, sometimes it may need to be redefined according to the requirement

in the derived class. An important feature of OOP that allows redefining a base class method in the derived class is overriding.

OOP's Concept

- Overriding is also referred to as run-time polymorphism (or late binding), as the compiler recognizes which specific implementation of method to invoke during the execution of program and thus, binds the appropriate method with the object at the run-time itself.

NOTES

3.10 KEY WORDS

- **Object:** A unit of structural and behavioural modularity that contains a set of properties (or data) as well as the associated functions.
- **Class:** A user-defined data type which contains the entire set of similar data and the functions that the objects possess.
- **Encapsulation:** A technique of binding or keeping together the data and the functions (that operate on them) in a single unit called a class.
- **Method:** A self-contained block of code that performs some function.
- **Inheritance:** A mechanism of deriving a new class (or derived class) from the old class (or base class) in such a way that the new class inherits all the members (variables and methods) of the old class
- **Polymorphism:** A feature of OOP that allows an entity (method or property) to be represented in various forms.
- **Overriding:** A feature of OOP that allows redefining a base class method in the derived class.

3.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Define the various features of OOP.
2. Why do we create classes and objects in OOP?
3. Define the term constructor and destructor.
4. What are events?
5. List the operators that can be overloaded, and that cannot be overloaded in VB.NET.
6. What do you mean by method overriding?

Long Answer Questions

1. Explain the concept of classes and objects in OOP. What is relationship between classes and objects?

NOTES

2. What is the use of creating properties in OOP? Explain its types.
3. Define the term destructor. What are the two types of destructors? Explain each with an example.
4. What do you mean by inheritance? How can we achieve inheritance in VB.NET? Explain with the help of an example.
5. What are the various types of inheritance? Explain with the help of suitable examples.
6. VB.NET does not support the feature of multiple inheritance directly. Is it true? If yes, what is the desired solution? Explain in brief.
7. What is operator overloading in .NET? Explain with the help of an example.
8. Differentiate between:
 - (a) Read-only and Write-only properties
 - (b) Base classes and Derived classes
 - (c) Compile-time polymorphism and Run-time polymorphism

3.12 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

BLOCK - II

VISUAL BASIC .NET

Introduction to VB.NET

UNIT 4 INTRODUCTION TO VB.NET

NOTES

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Structure and Programming Elements of VB.NET
 - 4.2.1 Keywords
 - 4.2.2 Data Types
 - 4.2.3 Variables
 - 4.2.4 Constants
 - 4.2.5 Operators
- 4.3 Arrays
- 4.4 String Handling
 - 4.4.1 Copy and Concatenating Strings
 - 4.4.2 Adding, Removing and Replacing Strings
 - 4.4.3 Formatting Strings
- 4.5 Answers to Check Your Progress Questions
- 4.6 Summary
- 4.7 Key Words
- 4.8 Self Assessment Questions and Exercises
- 4.9 Further Readings

4.0 INTRODUCTION

Visual Basic .NET is an object-oriented computer programming language that can be viewed as an evolution of Microsoft's Visual Basic (VB) which is generally implemented on the Microsoft .NET Framework. In any programming language, writing even an elementary program requires the knowledge and clear understanding of various data types, variables, constants and operators provided by that language. All these constitute the most basic elements of a language which are combined to form an instruction and a set of these instructions constitute a program. Generally, the instructions are executed in sequence in which they appear in the program.

4.1 OBJECTIVES

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

NOTES

- Study the structure of a VB.NET program
- Discuss various programming elements of VB.NET which include keywords, data types, variables, constants and operators
- Discuss an example program in VB.NET
- Declare, initialize and manipulate single-dimensional, multi-dimensional as well as dynamic arrays
- Understand and use string handling methods

4.2 STRUCTURE AND PROGRAMMING ELEMENTS OF VB.NET

Before discussing the programming elements of a VB.NET program, let us first discuss the general structure of a VB.NET program. A VB.NET program consists of the following parts:

- **Option** statement
- **Imports** statement
- **Namespace** declaration
- Class or Module declaration
- The **Main** procedure
- Variables, statements and expressions
- Comments

To understand the purpose of each part, consider a sample VB.NET code as shown in Figure 4.1.

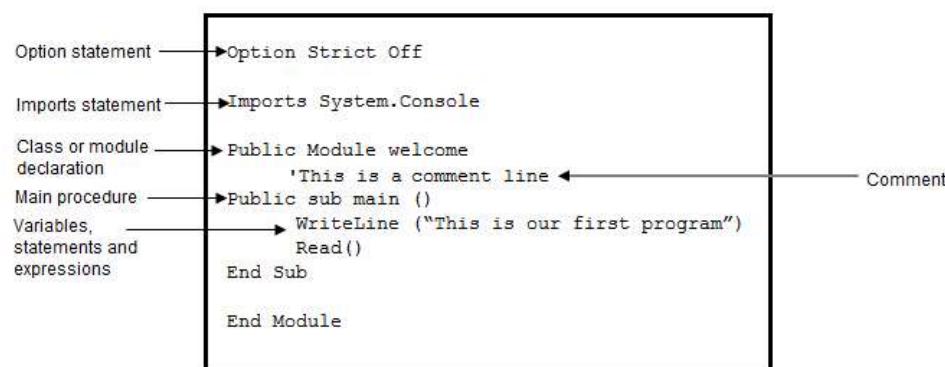


Fig. 4.1 Sample VB.NET Code

Option Statement

Option statement is the first statement that is to be written in the code but it is not always compulsory to write the option statement. Option statement helps prevent logic and syntax errors. There are four types of Option statements:

- **Option Explicit:** This statement has two values: On or Off. By default its value is On. It ensures that all the variables used in the program are declared before they are used in the program.
- **Option Compare:** This statement also has two values: Binary or Text. The default value is Binary. This statement specifies how string comparisons are made depending on the binary or text values.
- **Option Strict:** This statement prevents the occurrence of any kind of syntactical or logical errors or any kind of data loss that can occur when you work between variables of different types. It can be Off or On however, its default value is On.
- **Option Infer:** It also has the values On and Off. Default value is On for the projects that are created in Visual Basic 2010 whereas, default value is Off for the projects that are upgraded to Visual Basic 2010 from earlier versions.

NOTES

Imports Statement

Imports statement imports one or more namespaces in an application. It allows the code to refer to classes and other types defined within the namespace that is being imported, without the need of qualifying them. You can use as many Imports statements in your program. For example, in order to use the ReadLine method, you should first import the namespace System.Console. Here, Console is a class defined in the System namespace, and ReadLine is a method of the Console class.

Namespaces help to organize and classify your programming elements such as variables, classes, structures, etc., in such a way that they can be easily accessed in other applications. Namespaces prevent ambiguity and simplify references to avoid any naming conflicts between the programming elements classes that have the same name. For example, two classes with the same name can be used in an application if they belong to different namespaces. Some of the commonly used namespaces are given in Table 4.1

Table 4.1 Namespaces in Visual Basic 2010**NOTES**

Namespace	Description
System	It includes all the important classes and the base classes that define the data types, interfaces, attributes, exceptions, and event handlers.
System.Collections	It includes all those interfaces and classes that define collection of objects such as lists, queues, hash tables, arrays, and dictionaries.
System.Data	It provides access to classes that represent the ADO.NET architecture. ADO.NET lets you build components that efficiently manage data from multiple data sources.
System.Data.OleDb	It contains classes that support OLE DB.NET data provider.
System.Data.SqlClient	It contains classes that support SQL Server.NET data provider.
System.Data.OracleClient	It is the .NET Framework Data Provider for Oracle.
System.Data.Odbc	It is the .NET Framework Data Provider for ODBC.
System.Data.Linq	It contains classes that support interaction with relational databases in LINQ to SQL applications.
System.Drawing	It gives access to the GDI+ graphics packages that provides access to drawing methods.
System.Drawing.Printing	It comprises classes that help in customizing and performing the printing operations.
System.IO	It includes types that help in synchronous and asynchronous reading from and writing to data streams and files.
System.NET	It provides interface to the various protocols used on the Internet.
System.Security	It contains those classes that support the structure of the common language runtime security system.
System.Threading	It contains those classes and interfaces that enable multithreaded programming.
System.Web.Security	It contains classes that provide ASP.NET security in Web server applications.
System.Xml	It contains classes that support XML processing.

Note that once you have imported a namespace in your program, you can directly refer to the classes, functions, or other elements defined in that namespace without qualifying it with the namespace. However, if you do not import the namespace, you need to qualify the elements defined in the namespace each time you use them in the program as shown in the following statements:

```
System.Console.ReadLine()
System.Console.WriteLine("Hello World")
System.Console.Read()
```

In addition to System namespace, users can also define their own namespaces as follows:

```
Namespace myspace
    Structure mystruct      'defining a structure inside
        the namespace
        Public i As Integer
        Public str As String
    End Structure
End Namespace
```

Every program in VB.NET is completely object-oriented, so the program must contain a module or a class that contains the data and procedure used in your program.

NOTES**Main Procedure**

It is an essential part of a VB.NET program as it contains the `Main` method which is the starting point of a program. The `Main` procedure states what the module or class will do when executed. Once all the instructions in the `Main` method are executed, the control is transferred out of the class thus terminating the entire program. There are four varieties of `Main`:

- `Sub Main()`
- `Sub Main(ByVal CmdArgs() As String)`
- `Function Main() As Integer`
- `Function Main(ByVal CmdArgs() As String) As Integer`

`Sub Main()` is the most commonly used `Main`.

Variables, Statements, and Expressions

Inside the `Main` module, the variables are declared, input/output statements, and mathematical expressions are given. These describe the functionality of the module.

Comments

Comments are a vital element of a program. Comments are not executable statements and hence, do not increase the size of a file. Adding comments to the code makes it easier to understand the code. They act as a documentation that helps in understanding and maintaining existing code. In Visual Basic, comments start with an apostrophe (''). Visual Basic does not execute any statement followed by an apostrophe.

Programming Elements of VB.NET

In any programming language, writing even an elementary program requires the knowledge and clear understanding of various keywords, data types, variables, constants and operators provided by that language. All these constitute the most basic elements of a language which are combined to form an instruction and a set of these instructions constitute a program. This section discusses the basic elements of VB.NET.

4.2.1 Keywords

Keywords are the predefined words that have special significance in any language. Every keyword is reserved for a specific purpose and hence must not be used as user-defined names (identifiers). Some of the commonly used keywords in VB.NET are listed in Table 4.2.

NOTES

Table 4.2 Reserved Keywords in VB.NET

Alias	And	AndAlso	As	Boolean
Byte	ByVal	Call	Case	Catch
Char	Const	Continue	CShort	Date
Decimal	Declare	Default	Dim	Do
Double	Each	Else	ElseIf	End
EndIf	Enum	Erase	Event	Exit
Error	False	Finally	Friend	Function
Get	GetType	Goto	Handles	If
Implements	In	Imports	In	Integer
Is	IsNot	Like	Long	Loop
Me	Mod	Module	MyBase	MyClass
Namespace	New	Next	Not	Object
Of	On	Operator	Option	Or
Private	Protected	Public	RaiseEvent	ReadOnly
ReDim	Resume	Return	Select	Set
Short	Single	Static	Stop	Structure
Then	Throw	True	To	Try
Using	While	When	Wend	With

4.2.2 Data Types

A **data type** determines the type and the operations that can be performed on the data. VB.NET provides various data types and each data type is represented differently within the computer's memory. The type of data selected by a programmer depends on the particular application. The various data types provided by Visual Basic are broadly categorized into two types: *primitive* and *non-primitive*.

Primitive Data Types

Primitive data types, also known as **built-in data types**, are the fundamental data types provided by a programming language. Primitive data types available in VB.NET are listed in Table 4.3.

Table 4.3 Data Types in VB.NET

Type	Size (in bytes)	Range
Boolean	1	True or False
Byte	1	0 to 255(unsigned)
SByte	1	-128 to 127 (signed)
Char	2	0 to 65535(unsigned)
Short	2	-32,768 to 32767 (signed)
UShort	2	0 to 65535 (unsigned)
Integer	4	-2,147,483,648 to 2,147,483,647 (signed)
UInteger	4	0 to 4,294,967,295 (unsigned)
Long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (signed)
ULong	8	0 to 18,446,744,073,709,551,615 (unsigned)
Single (Single-precision floating-point)	4	3.4028235E+38 to -1.401298E-45 (for negative values) 1.401298E-45 to 3.4028235E+38 (for positive values)

Double (Double-precision floating-point)	8	-1.79769313486231570E+308 to -4.94065645841246544E-324 (for negative values) 4.94065645841246544E-324 to 1.79769313486231570E+308 (for positive values)
Decimal	16	0 to +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 to +/-7.9228162514264337593543950335 with 28 places to the right of the decimal smallest nonzero number is: +/-1E-28, that is, +/-0.00000000000000000000000000000001
Currency	8	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Date	8	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
Object	4	Any embedded object
Variant(numeric)	16	Any value as large as Double
Variant(text)	Length + 22 bytes	Same as variable-length string

NOTES

Non-Primitive Data Types

Non-primitive data types (user-defined data types) also known as **reference types** are derived from the primitive data types. In Visual Basic, these include *classes, structures, enumeration, interface, delegate, and arrays*.

4.2.3 Variables

A **variable** is an identifier that represents a memory location which is used to store data value. Data stored at a particular location can be accessed using the variable name. The value of a variable can be changed anytime during the program execution. The variable name we choose must be meaningful so as to understand what it represents in the program.

Declaring Variables

Like any other programming language, VB.NET also demands the declaration of variables before using it in the program. Variables in VB.NET are declared in the general section of the codes window using the `Dim` keyword, where `Dim` stands for dimension. It is a keyword which tells VB that a variable is being declared. This statement is used at the module, class, structure, procedure, or block level. The declaration of a variable informs the compiler, the specific data type to which a variable is associated and allocates sufficient memory for it.

The syntax for declaring a variable is:

```
Dim Variable_Name As Data_Type
```

For example, consider the following statements.

```

Dim firstname As String
Dim result As Integer
Dim newDate As Date
Dim firstname As String, Dim lastname As String

```

NOTES

The string variables can be declared as variable-length or fixed-length. The syntax given above is used for variable-length string. The fixed-length string is declared as follows:

```
Dim Variable Name As String * n
```

where, n is the number of characters a string can hold.

There are some important rules for naming variables:

- It must start with a letter not with a digit, and can be followed by letters and digits.
- It cannot be more than 255 characters.
- Spacing and period (such as, - and .) are not allowed.
- Reserved words cannot be used as name of variable.
- Special characters (such as, &, *, #) are not allowed.
- Uppercase and lowercase letters are not distinguishable in VB.

Some examples of valid and invalid variable names are shown in Table 4.4.

Table 4.4 Examples of Valid And Invalid Variable Names

Valid variables name	Invalid variables name
Emp_name	Emp.name (a period is not allowed)
Emp123	123Emp (first character cannot be a digit)
E123code	E123@code (special character @ is not allowed)

Type Declaration Characters

For specifying the data type of a variable we can also use some special characters in place of data type. These special characters are known as **type declaration characters**. Table 4.5 gives a list of type declaration characters used in VB.NET.

Table 4.5 Type Declaration Characters

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

The syntax for declaring a variable data type with type declaration characters is given below.

```
Dim Variable_Name<type-declaration character>
```

For example, consider this statement.

Introduction to VB.NET

```
Dim Emp_name$
```

In this statement, a variable `Emp_name` of type `String` is declared. This method of declaration of variable is known as **implicit declaration**. It is important to note that there should be no space between the variable name and its type declaration character.

Note: If we skip the datatype part during the declaration of the variable then, the data type of the variable will be **variant**. A variable of variant data type can hold value of any data type except user defined data types and fixed length string.

NOTES

Assigning Values to Variables

Declaration of variables simply allocates memory for variables—it does not store any data in the variables. To store data in the variables, values need to be assigned to them. The syntax for assigning values to variables is:

```
Variable=Expression
```

The expression can be a string, a number, an integer, a mathematical value, or a Boolean value. For example, consider these statements.

```
Dim x as Integer  
x=10
```

Here, a variable `x` of the integer type is declared and the value `10` is assigned to it.

4.2.4 Constants

Constants store values that remain the same throughout the execution of a program. Constants are used to improve the readability of your code by giving names to constant values. They are basically used when the value they contain needs to be used frequently in a program. For example, `rate_of_interest` can be made constant in a program which computes simple and compound interest for a given principal amount and time period. In future, if the rate of interest gets changed, its value needs to be modified only in one statement; all other values are automatically modified.

Declaring a Constant

The syntax to declare a constant is:

```
Const Constant_Name as Data_Type = Value
```

For example, consider the following statement.

```
Const PI As Single = 3.14
```

Compile-Time and Run-Time Constants

A **compile-time constant** is a value that is computed at compile-time. A **run-time constant** is a value that is computed while the program is running. If a same program runs more than once then a compile-time constant will have the same value each time the application is executed, while a run-time constant can have a

different value on each execution of the application.

4.2.5 Operators

NOTES

Operators are the symbols which perform operations on various data items. The data items on which the operators perform operations are known as **operands**. To perform an operation, operators and operands are combined together to form an **expression**. For example, $a + b$ is an expression, in which a and b are operands and $+$ is an operator.

Depending on the function performed, the operators in Visual Basic are classified into various categories. These include *arithmetic operators*, *assignment operators*, *comparison operators*, *string concatenation*, *relational operators*, *logical and bitwise operators* and *special operators*.

Arithmetic Operators

Arithmetic operators perform the basic arithmetic operations on operands. They can work on any built-in data type except Boolean. VB.NET provides various arithmetic operators which are listed in Table 4.6.

Table 4.6 Arithmetic Operators

Operator	Description
$+$	Addition or unary plus
$-$	Subtraction or unary minus
$*$	Multiplication
$/$	Division
\backslash	Integer division
$^$	Exponentiation
Mod	Modulus

Assignment Operators

Assignment operator assigns the value of an expression to a variable. Various assignment operators available in VB.NET are listed in Table 4.7.

Table 4.7 Assignment Operators

Operator	Description
$=$	Assignment
$^{=}$	Exponentiation followed by assignment
$*=$	Multiplication followed by assignment
$/=$	Division followed by assignment
$\backslash=$	Integer division followed by assignment
$+=$	Addition followed by assignment
$-=$	Subtraction followed by assignment
$\&=$	Concatenation followed by assignment
$<<=$	Perform arithmetic left shift on value of variable
$>>=$	Perform arithmetic right shift on value of variable

Comparison Operators

Comparison operators are used for comparing two values or expressions. They return values of Boolean type, that is, either true or false. Some of the various comparison operators provided by VB.NET are listed in Table 4.8.

Table 4.8 Comparison Operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
<>	Not equal to
Is	True if two object reference variables refer to the same object (irrespective of their values)
IsNot	True if two object references refer to the different objects
Like	Performs string matching against a pattern

NOTES

String Concatenation Operators

Concatenation operators join multiple strings into a single string. There are two concatenation operators in VB.NET, & and +. Both carry out the basic concatenation operation. The + operator can also concatenate numeric operands in addition to string operands, whereas, the & operator is defined only for string operands. In general, the & operator is recommended for string concatenation because it is defined exclusively for strings and minimizes the chances of generating an unintended conversion.

Logical and Bitwise Operators

Logical operators combine expressions containing comparison or logical operators, resulting into **logical expression**. The result of a logical expression can be either True or False. The operators which manipulate data at bit level are called bitwise operators. Various logical and bitwise operators provided by VB.NET are listed in Table 4.9.

Table 4.9 Logical and Bitwise Operators

Operator	Description
And	It returns True only if both operands are True and if even one of the operand is False, it returns False. It does the same for bit-by-bit operations where 0 is treated as False and 1 as True. Thus, for all 1s it gives 1, otherwise it returns 0.
Not	It performs an operation called inversion or complementation . It simply changes one logic level to the opposite, that is, True to False and vice versa; and 1 to 0 and 0 to 1 for bitwise operations.

NOTES

Or	It returns <code>true</code> if either operand is <code>True</code> for logical operations and for bit-by-bit operations it returns 1 if either of the operand is 1, otherwise it returns 0.
Xor	This operator performs an exclusive-Or operation. For logical operations, it returns <code>True</code> if there are odd number of <code>True</code> in the expression; otherwise it returns <code>False</code> . The same applies for bit-by-bit operations where it returns 1 for off number of 1s, and 0 otherwise.
AndAlso	This operator checks the first operand, if it is <code>False</code> , the second operand is not tested.
OrElse	This operator checks the first operand, if it is <code>True</code> , the second operand is not tested.
IsTrue	Checks whether the operation is <code>True</code> .
IsFalse	Checks whether the operation is <code>False</code> .

Operators Precedence and Associativity

An expression consisting of more than one operator leads to a confusion as to which operator is to be evaluated first. For example, consider this expression.

`a + b * c - d`

In this expression, the compiler needs to know which operator is evaluated first. For this, it is important to determine the precedence and associativity of operators.

- **Precedence:** The order or priority in which various operators in an expression are evaluated is known as **precedence**. Every operator in VB.NET has a precedence associated with it. The operators with a higher precedence are evaluated before the operators with a lower precedence. For example, multiplication is performed before addition as the multiplication operator has higher precedence than the addition operator.
- **Associativity:** The order or priority in which operators of the same precedence are evaluated is known as **associativity**. For example, the addition and subtraction operators have the same precedence. The compiler evaluates each operation as it encounters it from left to right.

The precedence of operators in VB is shown as follows:

- Exponentiation (^)
- Unary identity and negation (+, -)
- Multiplication and floating-point division (*, /)
- Integer division (\)
- Modulus arithmetic (Mod)
- Addition and subtraction (+, -)
- All comparison operators (=, <>, <, <=, >, >=, Is, IsNot, etc.)
- Negation (Not)

- Conjunction (And, AndAlso)
- Inclusive disjunction (Or, OrElse)
- Exclusive disjunction (Xor)

Introduction to VB.NET

An Example VB Program

To understand the use of the programming elements of VB.NET discussed so far, consider an example VB.NET program given in Example 4.1. This program is a console application as it displays the output in the command prompt window. To create a VB.NET console application, click **New Project** from the **Start page** which appears when you start Visual Studio 2010, or click **New** from the **File** menu in the Visual Studio 2010 IDE. This displays the New Project window. Choose **Console Application**, and click **OK** (see Figure 4.2). A module named *Module1.vb* is created, where you can type the VB.NET code.

NOTES

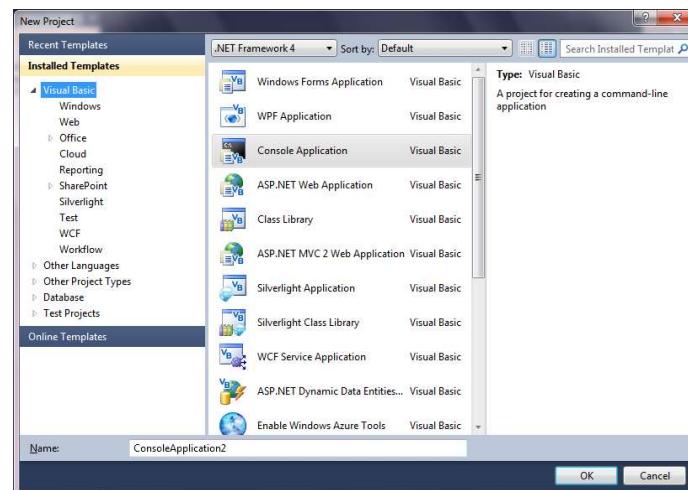


Fig. 4.2 Creating a Console Application

Example 4.1: A program to demonstrate the use of programming elements of VB.NET.

```

Imports System.Console
Module Module1
    Sub Main()
        Dim firstname$, lastname$
        Dim age As Integer
        Dim marks, percentage As Single
        Const n As Integer = 5
        WriteLine("Enter the first name")
        firstname = ReadLine() 'input the firstname from
        the user
        WriteLine("Enter the last name")
        lastname = ReadLine() 'input the lastname from
    End Sub

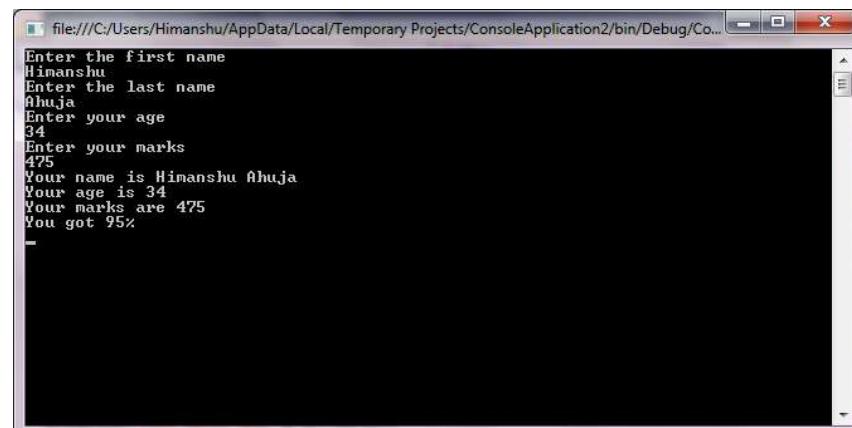
```

```

        WriteLine("Enter your age")
        age = ReadLine() 'age is read as string not integer
        WriteLine("Enter your marks")
        marks = Integer.Parse(ReadLine())
        'Integer.Parse is used to convert String to Integer
        percentage = marks / n
        WriteLine("Your name is " & firstname & " " &
lastname)
        WriteLine("Your age is" & Str(age))
        WriteLine("Your marks are" & Str(marks))
        WriteLine("You got" & Str(percentage) & "%")
        Read()
    End Sub
End Module

```

The output of the program is



4.3 ARRAYS

As discussed, Visual Basic uses variables of different built-in data types to store data. However, these variables are incapable of holding more than one value at a time. For example, a single variable cannot be used for storing the names of all the students in a class. For such purposes, Visual Basic provides a different kind of data type known as arrays.

Arrays are defined as a fixed size sequence of same type of data elements. These data elements can be of any built-in or user-defined data type. The elements of an array are stored in contiguous memory locations and each individual element can be accessed using one or more *indices*. An **index** is a positive integer value, which indicates the position of an element in an array. Arrays are used in situations where a programmer wants to store a list of data items into a single variable and

also wants to access and manipulate individual elements of the list. Arrays can be either single-dimensional or multi-dimensional depending upon the number of subscripts used.

In VB.NET, all the arrays are inherited from the `System.Array` class. The `Array` class is a member of the `System` namespace. The `Array` class provides methods for creating, searching, sorting, and modifying arrays. Some of the commonly used methods of the `Array` class are `GetUpperBound`, `GetLowerBound`, and `GetLength`.

NOTES

Single-Dimensional Arrays

A single-dimensional array is the simplest form of an array that requires only *one* subscript to access an array element. Like an ordinary variable, an array must have been declared before it is used in the program. The syntax for declaring a single-dimensional array is:

```
Dim Array_Name (Num_Elements) As Data_Type
```

where

`Array_Name` refers to the name of the array.

`Num_Elements` refers to the number of elements the array can contain.

`Data_Type` refers to the data type of elements. This parameter is optional.

For example, an array `marks` of type `Integer` and size 5 can be declared using this statement.

```
Dim marks (5) As Integer
```

Initialization of Single-Dimensional Array

Once an array is declared, the next step is to initialize each array element with a valid and appropriate value. Note that unless an array is initialized, all the array elements contain garbage values. An array can be initialized at the time of its declaration.

The syntax for initializing an array at the time of its declaration is:

```
Dim Array_Name () As Data_Type = {value 1,value  
2,.....,value n}
```

The values are assigned to the array elements in the order in which they are listed. That is, `value 1`, `value 2`, ..., `value n` are assigned to the first, second, ..., nth element of the array, respectively.

For example, the array `marks` can be initialized while declaring using this statement.

```
Dim marks () As Integer={51,62,43,74,55}
```

Example 4.2: A program to demonstrate the use of single-dimensional arrays.

```
Imports System.Console  
Module Module1
```

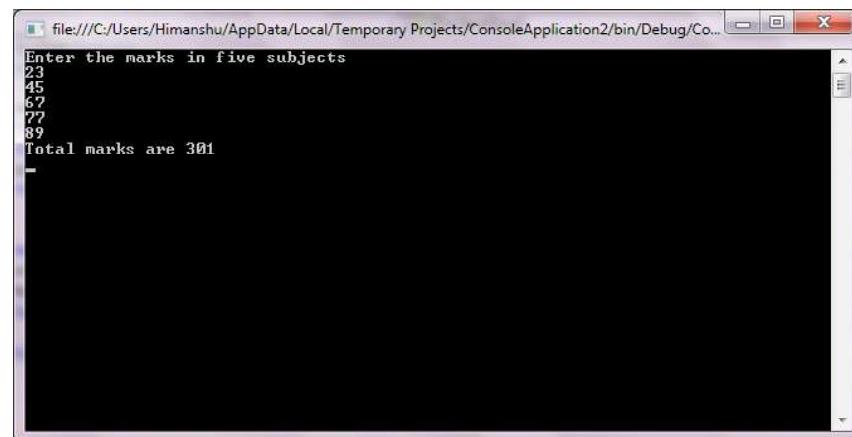
NOTES

```

Sub main()
    Dim marks(5) As Integer
    Dim sum As Integer
    WriteLine("Enter the marks in five subjects")
    For i As Integer = 0 To 4
        marks(i) = Integer.Parse(ReadLine())
        sum += marks(i)
    Next
    WriteLine("Total marks are" & Str(sum))
    Read()
End Sub
End Module

```

The output of the program is



Note: The programs of arrays make use of loops which are discussed later in this section.

Multi-Dimensional Arrays

A multi-dimensional array of dimension n is a collection of items that are accessed with the help of n subscript values. Generally, the arrays of three or more dimensions are not used because of their huge memory requirements and the complexities involved in their manipulation. Hence, only two-dimensional arrays will be discussed in this unit.

Two-dimensional array

A two-dimensional array is the simplest form of a multi-dimensional array that requires *two* subscript values to access an array element. Two-dimensional arrays are useful when data being processed can be arranged in the form of rows and columns (matrix form). The syntax for declaring a two-dimensional array is:

```
Dim Array_Name(row_size, column_size) As Data_Type
```

For example, an array a of type Integer having three rows and two columns can be declared using this statement.

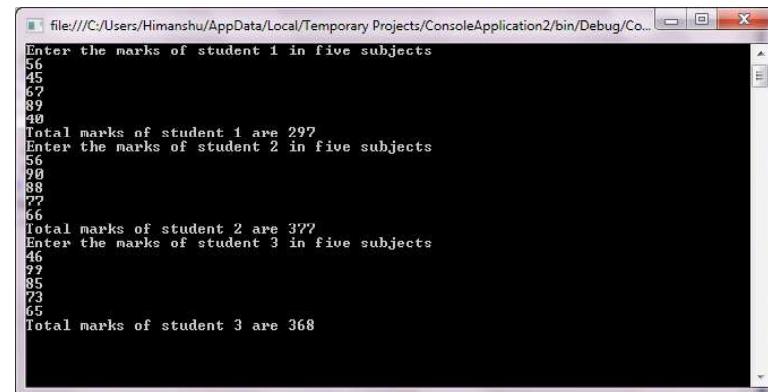
```
Dim a(3, 2) As Integer
```

Example 4.3: A program to demonstrate the use of two-dimensional arrays.

Introduction to VB.NET

```
Imports System.Console
Module Module1
    Sub main()
        Dim stu_marks(3, 5) As Integer
        Dim sum As Integer
        For i As Integer = 0 To 2
            WriteLine("Enter the marks of student" & Str(i + 1) & " in five subjects")
            For j As Integer = 0 To 4
                stu_marks(i, j) = Integer.Parse(ReadLine())
                sum += stu_marks(i, j)
            Next j
            WriteLine("Total marks of student" & Str(i+1) & " are" & Str(sum))
            sum = 0
        Next i
        Read()
    End Sub
End Module
```

The output of the program is



NOTES

Dynamic Arrays

Sometimes you come across situations in which you do not know the number of elements to be stored in an array. For example, the number of candidates applying for a job is not known in advance, so it is not possible to specify the size of such an array in advance. For such kind of situations dynamic arrays are used. The size of a dynamic array can vary during the execution of a program. ReDim statement can be used to specify or change the size of one or more dimensions of an already declared array. Some of the features of the ReDim statement are as follows:

NOTES

- It initializes the elements of the new array with the default values of their data type. It does not change the data type of the array or initialize new values for the array elements.
- It cannot be used at class or module level. It can be used only at the procedure level.

The syntax of ReDim statement is:

```
ReDim [Preserve] varname (subscripts)
```

where

subscripts specifies the new dimension

Preserve Keyword

There is a risk of losing the data when the size of array is changed. The Preserve keyword preserves the data in an existing array. This is done by copying the elements of the old array to the new one before modifying the dimension of the array. For example, consider these statements.

```
Dim a() As Integer = {1, 2}
```

```
ReDim Preserve a(10)
```

Example 4.4: A program to demonstrate the use of dynamic arrays.

```
Imports System.Console
Module Module1
    Sub main()
        Dim stu_marks(3) As Integer
        Dim i As Integer
        For i = 0 To 2
            WriteLine("Enter the total marks of student" &
Str(i + 1))
            stu_marks(i) = Integer.Parse(ReadLine())
        Next
        ReDim Preserve stu_marks(6) 'redefining the array
        For i = 3 To 5
            WriteLine("Enter the total marks of student" &
Str(i + 1))
            stu_marks(i) = Integer.Parse(ReadLine())
        Next
        ReDim Preserve stu_marks(10)
        For i = 6 To 9
            WriteLine("Enter the total marks of student" &
Str(i + 1))
            stu_marks(i) = Integer.Parse(ReadLine())
        Next
    End Sub
End Module
```

```

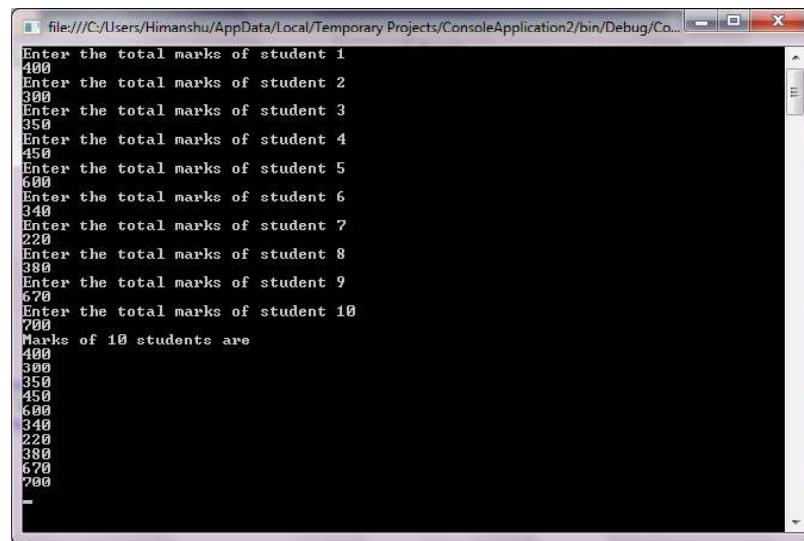
    Next
    WriteLine("Marks of 10 students are")
    For i = 0 To 9
        WriteLine(stu_marks(i))
    Next
    Read()
End Sub
End Module

```

Introduction to VB.NET

NOTES

The output of the program is



4.4 STRING HANDLING

There are classes that perform the task of checking or comparing the two strings. The return result always comes in true or false. The compare result comes in number, which can be less than zero or greater than zero. The result is stored in the integer data type.

Value Meaning

- **When the result is less than zero:** This means that the first string is less than second.
- **When the result is zero:** This means that both strings are equal.
- **When the result is greater than zero:** This means that the first string is greater than zero.

The following example compares two strings:

```

Dim str1 As String = "radix"
Dim str2 As String = "solution"
Dim res As Int16 = String.Compare(str1, str2)

```

NOTES

```

        Console.WriteLine("First result:" +
res.ToString())
        str2 = "ttt"
        res = String.Compare(str1, str2)
        Console.WriteLine("Second result:" +
res.ToString())
        str1 = "ttt"
        res = String.Compare(str1, str2)
        Console.WriteLine("Third result:" +
res.ToString())

```

The `CompareTo` method is an instance method. It compares a value with a string instance. The following source code compares two strings:

```

' CompareTo Method
Dim str As String = "radix"
Console.WriteLine(str.CompareTo(str1))

```

4.4.1 Copy and Concatenating Strings

The `Concat` method adds string (or objects) and returns a new string. Using `Concat` method, you can add two strings, two objects and one string, one object or more combinations of these two.

The following source code concatenates two strings:

```

Dim str1 As String = "ppp"
Dim str2 As String = "ccc"
Dim strRes As String = String.Concat(str1, str2)
Console.WriteLine(strRes)

```

The following source code concatenates one string and one object:

```

Dim obj As Object = 12
strRes = String.Concat(str1, obj)
Console.WriteLine(strRes)

```

The `Copy` method copies contents of a string to another. The `Copy` method takes a string as input and returns another string with the same content as the input string.

For example, the following code copies `str1` to `strRes`:

```

strRes = String.Copy(str1)
Console.WriteLine("Copy result :" + strRes)

```

The `CopyTo` method copies a specified number of characters to a specified position. Consider the following example:

```

Dim str1 As String = "pp"
Dim chrs(2) As Char

```

```
str1.CopyTo(0, chrs, 0, 2)
Console.WriteLine(chrs(0) + chrs(1))
```

Introduction to VB.NET

The `Clone` method returns a new copy of a string in the form of an object. The following code creates a clone of `str1`.

```
Dim str1 As String = "ppp"
Dim objClone As Object = str1.Clone()
Console.WriteLine("Clone :" +
objClone.ToString())
```

The `Join` method is useful when you need to insert a separator (String) between each element of a string array, yielding a single concatenated string. For example, the following sample inserts a comma and space (', ') between each element of an array of strings:

```
Dim str1 As String = "ppp"
Dim str2 As String = "ccc"
Dim str3 As String = "kkk"
Dim allStr() As String = New String() {str1,
str2, str3}
Dim strRes As String = String.Join(", ", allStr)
Console.WriteLine("Join Results: " + strRes)
```

4.4.2 Adding, Removing and Replacing Strings

The `Insert` method inserts a specified string at a specified index position in an instance. The following source code inserts 'rsd' after second character in `str1` and the result string is 'samp':

```
Dim str1 As String = "rsd"
Dim strRes As String = str1.Insert(2, "samp")
Console.WriteLine(strRes.ToString())
```

The `Remove` method deletes a specified number of characters from a specified position in a string. This method returns result as a string. For example, the following code removes three characters from index 3:

```
Dim s As String = "123abc000"
Console.WriteLine(s.Remove(3, 3))
```

The `Replace` method replaces all occurrences of a specified character in a string. For example, the following source code replaces all p character instances of `str1` with character l and returns string 'lll':

```
Dim str1 As String = "ppp"
Dim repStr As String = str1.Replace("p", "l")
Console.WriteLine("Replaced string:" +
repStr.ToString())
```

The `Split` method separate strings by a specified set of characters and places these strings into an array of strings. For example, the following source code splits

NOTES

NOTES

`strArray` based on ',' and stores all separated strings in an array.

```
Dim str1 As String = "ppp"
Dim str2 As String = "ccc"
Dim str3 As String = "kkk"
Dim strAll3 As String = str1 + ", " + str2 + ", "
+ str3
Dim strArray() As String = strAll3.Split(",")
```

Uppercase and Lowercase

The `ToUpper` and `ToLower` methods convert a string in uppercase and lowercase. These methods are easy to implement. The following code shows how to use `ToUpper` and `ToLower` methods:

```
Dim aStr As String = "adgas"
Dim bStr As String = "ABNMDWER"
Dim strRes As String = aStr.ToUpper()
Console.WriteLine("Uppercase:" + strRes.ToString())
strRes = bStr.ToLower()
Console.WriteLine("Lowercase:" + strRes.ToString())
```

4.4.3 Formatting Strings

You can use the `Format` method to create formatted strings and concatenate multiple strings representing multiple objects. The `Format` method automatically converts any passed object into a string. For example, the following code uses integer, floating number and string values and format them into a string using the `Format` method:

```
Dim val As Int16 = 7
Dim name As String = "Mr. John"
Dim num As Double = 45.06F
Dim str As String = String.Format ("Days Left: {0}. Current DateTime: {1: u}. \n String: {2}, Float: {3}", val, DateTime.Now, name, num)
Console.WriteLine(str)
```

Trimming and Removing Characters from Strings

The `String` class provides `Trim`, `TrimStart` and `TrimEnd` methods to trim strings. The `Trim` method removes white spaces from the beginning and end of a string. The `TrimEnd` method removes characters specified in an array of characters from the end of a string and `TrimStart` method removes characters specified in an array of characters from the beginning of a string. You can also use

the Remove method to remove characters from a string. The following code shows how to use these methods:

```
Dim str As String = " C# "
Console.WriteLine("Hello{0}World!", str)
Dim trStr As String = str.Trim()
Console.WriteLine("Hello{0}World!", trStr)
str = "Hello World!"
Dim chArr() As Char = {"e", "H", "l", "o", " "}
trStr = str.TrimStart(chArr)
Console.WriteLine(trStr)
str = "Hello World!"
Dim chArr1() As Char = {"e", "H", "l", "o", " "}
trStr = str.TrimEnd(chArr1)
Console.WriteLine(trStr)
Dim MyString As String = "Hello Delta World!"
Console.WriteLine(MyString.Remove(5, 10))
```

Introduction to VB.NET

NOTES

Padding Strings

The PadLeft and PadRight methods can be used to pad strings. The PadLeft method right-aligns and pads a string so that its rightmost character is the specified distance from the beginning of the string. The PadRight method left-aligns and pads a string so that its rightmost character is a specified distance from the end of the string. These methods return new String objects that can either be padded with empty spaces or with custom characters. The following code shows how to use these methods:

```
Dim str1 As String = "My String"
Console.WriteLine(str1.PadLeft(20, "-"))
Dim str2 As String = "My String"
Console.WriteLine(str2.PadRight(20, "-"))
```

Using StringBuilder Class

The `StringBuilder` class represents a mutable string of characters. It is known as mutable because it can be modified once it has been created with the help of `Append`, `Insert`, `Remove`, and `Replace` methods.

Note: When multiple concatenation operations are involved with strings, for performance reasons, it is recommended to use `StringBuilder` instead of `String`.

The `StringBuilder` class is defined in the `System.Text` namespace. Before you use the `StringBuilder` class, make sure you add imports `System.Text` line in your application.

NOTES

Check Your Progress

1. Name the parts comprised by a VB.NET program.
2. What is main procedure? What are the four varieties of main?
3. What are primitive and non-primitive data types?
4. What are constants? What is the syntax to declare a constant in VB.NET?
5. What is a single dimensional array?
6. What is the use of Concat method?

4.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A VB.NET program consists of the following parts:
 - (i) Option statement
 - (ii) Imports statement
 - (iii) Namespace declaration
 - (iv) Class or Module declaration
 - (v) The Main procedure
 - (vi) Variables, statements and expressions
 - (vii) Comments
2. Main is an essential part of a VB.NET program as it contains the Main method which is the starting point of a program. The Main procedure states what the module or class will do when executed. Once all the instructions in the Main method are executed, the control is transferred out of the class thus terminating the entire program. There are four varieties of Main:
 - (i) Sub Main()
 - (ii) Sub Main(ByVal CmdArgs() As String)
 - (iii) Function Main() As Integer
 - (iv) Function Main(ByVal CmdArgs() As String)
As Integer
3. Primitive data types, also known as built-in data types, are the fundamental data types provided by a programming language. Non-primitive data types (user-defined data types) also known as reference types are derived from the primitive data types. In Visual Basic, these include classes, structures, enumeration, interface, delegate, and arrays.

4. Constants store values that remain the same throughout the execution of a program. Constants are used to improve the readability of your code by giving names to constant values. They are basically used when the value they contain needs to be used frequently in a program. For example, `rate_of_interest` can be made constant in a program which computes simple and compound interest for a given principal amount and time period. In future, if the rate of interest gets changed, its value needs to be modified only in one statement; all other values are automatically modified.

The syntax to declare a constant is:

```
Const Constant_Name as Data_Type = Value
```

5. A single-dimensional array is the simplest form of an array that requires only one subscript to access an array element.
6. The `Concat` method adds string (or objects) and returns a new string.

NOTES

4.6 SUMMARY

- A VB.NET program consists of an `Option` statement, `Imports` statement, Namespace declaration, Class or Module declaration, the `Main` procedure, variables, statements and expressions, and comments.
- `Option` statement is the first statement that is to be written in the code but it is not always compulsory to write the `Option` statement. `Option` statement helps prevent logic and syntax errors.
- Four types of option statements are `Option Explicit`, `Option Compare`, `Option Strict`, `Option Infer`.
- `Imports` statement imports one or more namespaces in an application. It allows the code to refer to classes and other types defined within the namespace that is being imported, without the need of qualifying them.
- Namespaces help to organize and classify your programming elements, such as variables, classes, structures, etc., in such a way that they can be easily accessed in other applications. Namespaces prevent ambiguity and simplify references to avoid any naming conflicts between the programming elements classes that have the same name.
- The `Main` procedure states what the module or class will do when executed. Once all the instructions in the `Main` method are executed, the control is transferred out of the class thus terminating the entire program.
- Comments are a vital element of a program. Comments are not executable statements and hence, do not increase the size of a file. Adding comments to the code makes it easier to understand the code. They act as a documentation that helps in understanding and maintaining existing code. In Visual Basic, comments start with an apostrophe (''). Visual Basic does not execute any statement followed by an apostrophe.

NOTES

- Keywords are the predefined words that have special significance in any language. Every keyword is reserved for a specific purpose and hence must not be used as user-defined names (identifiers).
- A data type determines the type and the operations that can be performed on the data. VB.NET provides various data types and each data type is represented differently within the computer's memory. The various data types provided by Visual Basic are broadly categorized into two types: primitive and non-primitive.
- Primitive data types, also known as built-in data types, are the fundamental data types provided by a programming language.
- Non-primitive data types (user-defined data types) also known as reference types are derived from the primitive data types. In Visual Basic, these include classes, structures, enumeration, interface, delegate, and arrays.
- A variable is an identifier that represents a memory location which is used to store data value. Data stored at a particular location can be accessed using the variable name. The value of a variable can be changed anytime during the program execution.
- Constants store values that remain the same throughout the execution of a program. Constants are used to improve the readability of your code by giving names to constant values. They are basically used when the value they contain needs to be used frequently in a program.
- Operators are the symbols which perform operations on various data items. The data items on which the operators perform operations are known as operands. To perform an operation, operators and operands are combined together to form an expression.
- Depending on the function performed, the operators in Visual Basic are classified into various categories. These include arithmetic operators, assignment operators, comparison operators, string concatenation, relational operators, logical and bitwise operators and special operators.
- Arrays are defined as a fixed size sequence of same type of data elements. These data elements can be of any built-in or user-defined data type. The elements of an array are stored in contiguous memory locations and each individual element can be accessed using one or more indices. An index is a positive integer value, which indicates the position of an element in an array.
- A single-dimensional array is the simplest form of an array that requires only one subscript to access an array element. Like an ordinary variable, an array must have been declared before it is used in the program.
- A multi-dimensional array of dimension n is a collection of items that are accessed with the help of n subscript values.
- A two-dimensional array is the simplest form of a multi-dimensional array that requires *two* subscript values to access an array element. Two-

dimensional arrays are useful when data being processed can be arranged in the form of rows and columns (matrix form).

- Dynamic arrays are used in the situations in which the number of elements to be stored in an array is not known in advance. The size of a dynamic array can vary during the execution of a program. ReDim statement can be used to specify or change the size of one or more dimensions of an already declared array.
- The Concat method adds string (or objects) and returns a new string.
- You can use the Format method to create formatted strings and concatenate multiple strings representing multiple objects. The Format method automatically converts any passed object into a string.

NOTES

4.7 KEY WORDS

- **Keywords:** The predefined words that have special significance in any language—every keyword is reserved for a specific purpose.
- **Data Type:** Determines the type and the operations that can be performed on the data.
- **Variable:** An identifier that represents a memory location which is used to store data value and the value of a variable can be changed anytime during the program execution.
- **Constants:** They store values that remain the same throughout the execution of a program.
- **Operators:** The symbols which perform operations on various data items.
- **Arrays:** A fixed-size sequence of same type of data elements and these data elements can be of any built-in or user-defined data type.

4.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Why the Imports statement used in VB.NET?
2. What do you understand by precedence and associativity of operators?
3. What are the logical and bitwise operators provided by VB.NET?
4. What do you understand by multidimensional arrays?

Long Answer Questions

1. What is data type? Discuss the various types of data types provided by VB.NET.

NOTES

2. Name and explain the different types of operators used in VB.NET.
3. What is an array? How are elements stored in an array? Discuss the various types of arrays.
4. What is a dynamic array? Also explain the use of `ReDim` and `Preserve` keywords in VB.
5. What are the two ways of passing parameters to a procedure? Explain with the help of suitable examples.

4.9 FURTHER READINGS

Holzner, Steven. 2002. *Visual Basic .NET Black Book*, United Kingdoms: Coriolis Group Books.

Deitel, Harvey M. 2001. *Visual Basic.NET How to Program*, 2nd edition. New Jersey: Prentice Hall.

UNIT 5 CONTROL STATEMENTS

Structure

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Conditional and Looping Statements
- 5.3 Procedures in VB.NET
 - 5.3.1 Sub Procedures
 - 5.3.2 Functions
 - 5.3.3 MsgBox() and InputBox() Functions
- 5.4 Answers to Check Your Progress Questions
- 5.5 Summary
- 5.6 Key Words
- 5.7 Self Assessment Questions and Exercises
- 5.8 Further Readings

NOTES

5.0 INTRODUCTION

To make a program more flexible and efficient, the flow of execution can be altered using various control statements. Different types of control flow statements, such as selection statements and iteration statements are discussed in this unit. This unit also discusses arrays, procedures, functions and two of the commonly used functions that is `MsgBox()` and `Input Box()`.

5.1 OBJECTIVES

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

- Understand the different types of control statements in VB.NET
- Explain the various types of selection statements
- Discuss the different types of iteration statements
- Learn how procedures and functions are created in VB.NET

5.2 CONDITIONAL AND LOOPING STATEMENTS

A statement is an instruction given to the computer to perform a specific action. By default, the statements are executed in the same order in which they appear in the program and each statement is executed only once. However, the serial execution of statements makes a program inflexible and unsuitable for most of the practical applications. To make a program more flexible, control statements are used to alter the flow of control of the program.

In VB.NET, the control statements are broadly classified into three categories, namely, *selection statements*, *iteration statements* and *jump*

NOTES

statements. All these control statements are commonly used with the logical tests or test conditions to alter the flow of control conditionally or unconditionally. To alter the flow conditionally, a particular condition is evaluated to control the flow of execution. On the other hand, to alter the flow unconditionally, no such condition is evaluated.

Selection Statements (Decision-Making)

Selection statements, also known as **conditional statements**, are used to make decisions based on a given condition. If the condition evaluates to True, a set of statements is executed, otherwise another set of statements is executed. Various types of selection statements supported by VB.NET are:

- If statement
- If-Else statement
- Nested If-Else statement
- Select...Case statement

The If Statement

The If statement selects and executes the statement(s) based on a given condition. If the condition evaluates to True, then a given set of statement(s) is executed. However, if the condition evaluates to False, then the given set of statements is skipped and the program control passes to the statement following the If statement (see Figure 5.1).

The syntax of the If statement is

```
If condition Then
    statements
End If
```

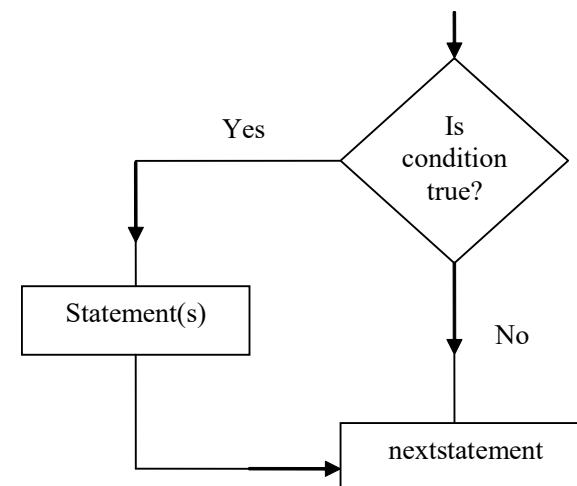


Fig. 5.1 Flow of Control in If Statement

The If-Else Statement

Control Statements

The If-Else statement causes one of the two possible statement(s) to execute depending upon the outcome of condition.

The syntax of the If-Else statements is:

```
If condition Then  
    statements  
Else  
    elsestatements  
End If
```

NOTES

Here, the If-Else statement comprises two parts, namely, If and Else. If the IF condition is True, then part is executed; otherwise, the Else part is executed (see Figure 5.2).

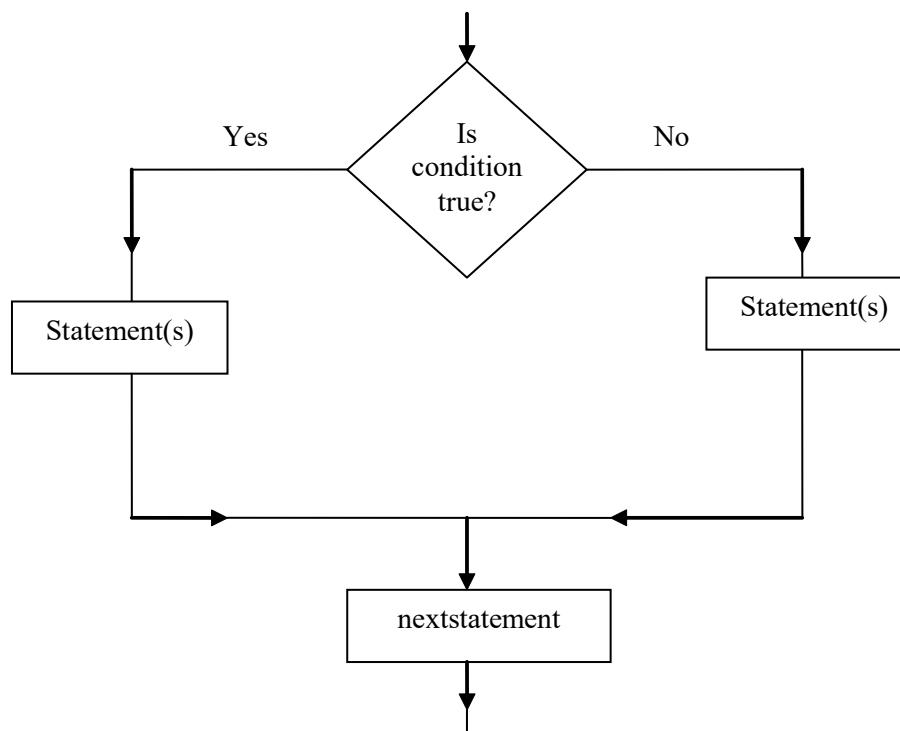


Fig. 5.2 Flow of Control in If-Else Statement

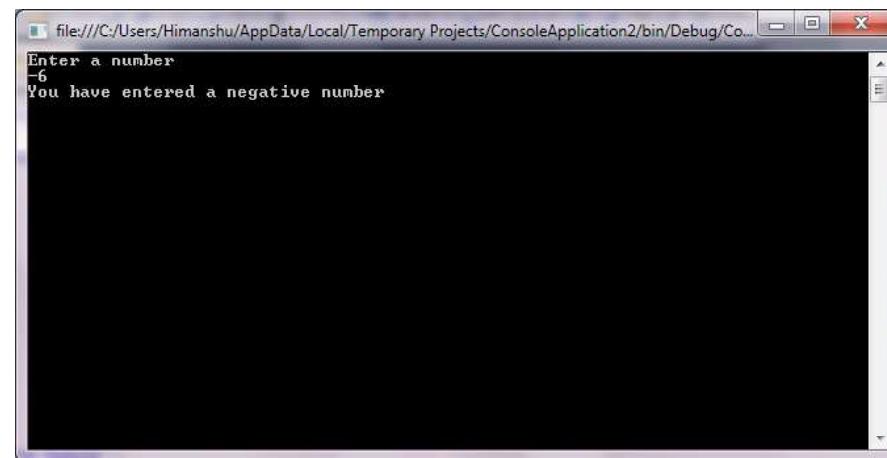
Example 5.1: A program to demonstrate the use of If-Else statement.

```
Imports System.Console  
Module Module1  
    Sub Main()  
        Dim i As Integer  
        WriteLine("Enter a number")  
        i = Integer.Parse(ReadLine())  
        If i > 0 Then
```

```

        WriteLine("You have entered a positive number")
    Else
        WriteLine("You have entered a negative number")
    End If
    Read()
End Sub
End Module

```

NOTES**The output of the program is****Nested If-Else Statement**

A nested If-Else statement contains one or more If-Else statements. The syntax of nested If-Else statement is:

```

If condition1 Then
    statement1
ElseIf condition2 Then
    statement2
Else
    statement3
End If

```

Example 5.2: A program to demonstrate the use of nested If-Else statement

```

Imports System.Console
Module Module1
    Sub Main()
        Dim marks As Integer
        WriteLine("Enter your marks")
        marks = Single.Parse(ReadLine())
        If marks < 33 Then
            WriteLine("You have failed")
        ElseIf marks > 33 And marks <= 50 Then

```

```

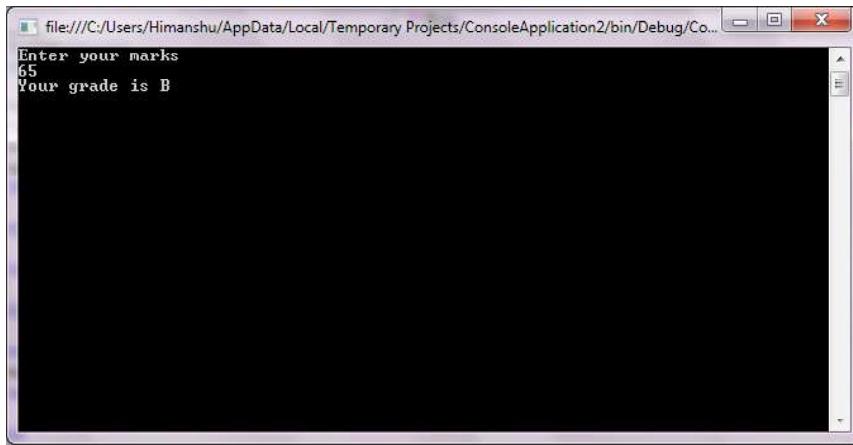
        WriteLine("Your grade is C")
ElseIf marks > 50 And marks <= 75 Then
    WriteLine("Your grade is B")
Else
    WriteLine("Your grade is A")
End If
Read()
End Sub
End Module

```

Control Statements

NOTES

The output of the program is



Selects...Case Statements

Select...Case in VB.NET is also a kind of selection statement. It is used in those situations where there are many choices and for those choices, more than one condition or case is required. The Select...Case statement makes use of a variable or an expression as a test variable. The flow of the program is controlled by this variable therefore, this variable is also known as **control variable**. Based on this variable an expression from the expression list is selected. The syntax of the Select...Case statement is:

```

Select (Case) testexpression
Case value
    statement1
Case value
    statement2
Case value
    statement3
Case Else
    statement4
End Select

```

NOTES

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

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

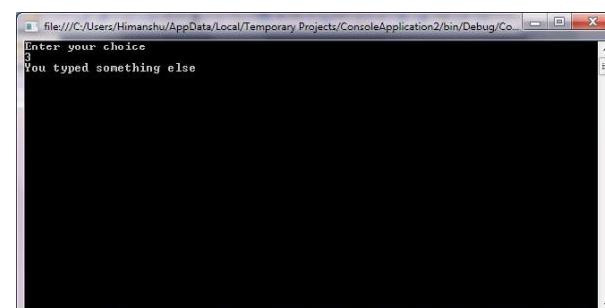
Example 5.3: A program to demonstrate the use of nested Select...Case statement.

```

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

```

The output of the program is



The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as **iteration statements**. That is, as long as the condition evaluates to True, the set of statement(s) is executed. The various iteration statements used in VB.NET are as follows:

- For statement
- For Each statement
- While loop
- Do loop

NOTES**The For Loop**

The For loop is one of the most widely used loops. The For loop is a deterministic loop in nature, that is, the number of times the body of the loop is executed is known in advance.

The syntax of the For loop is:

```
For counter (As datatype)=(start) To (end) Step (step)
    statements
    Next counter
```

Note that counter defines the numeric control variable for the loop, datatype defines the data type of the counter, initial value of the counter is given by start and final value of the counter is given by end. Step specifies the value by which counter is incremented or decremented every time. If Step clause is not used, the control variable is incremented by 1 by default. If we specify a negative value with the Step clause, the counter variable will be decremented with that value each time the loop is executed. Next will increment or decrement the counter variable for next iteration.

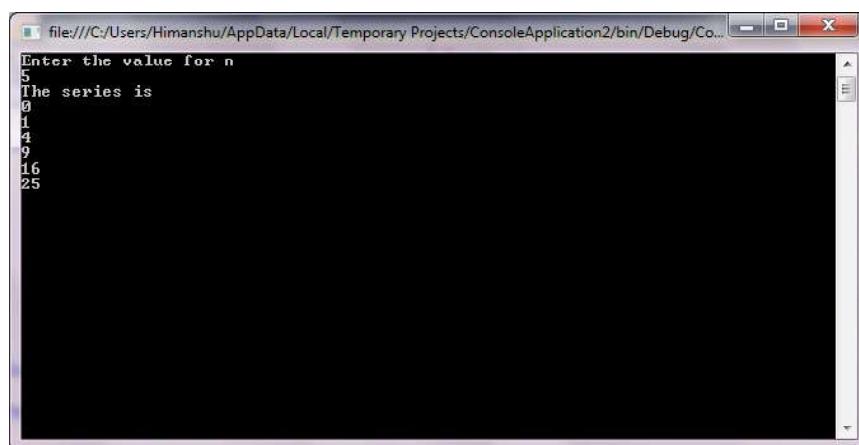
Example 5.4: A program to display a series using For loop.

```
Imports System.Console
Module Module1
    Sub Main()
        Dim n As Integer
        WriteLine("Enter the value for n")
        n = Integer.Parse(Console.ReadLine())
        'This loop goes from 0 to n.
        WriteLine("The series is ")
        For value As Integer = 0 To n
            Console.WriteLine(value * value)
        Next
```

```

        Read()
End Sub
End Module

```

NOTES**The output of the program is****The For Each Loop**

For Each loop works for a list that may be an array or a collection of objects. This loop executes iteratively through all the items in a list.

The syntax of the For Each statement is:

```

For Each element (As datatype) In group
statements
Next element

```

Note that element defines a variable which is used to iterate through elements of an array. The datatype defines the data type of the element variable, if it is not already declared. Group specifies the array or collection of objects over which the statements contained in the For Each block are to be repeated. Statements stand for the statements that are executed for given number of times. Next increment the element variable for the next iteration.

Example 5.5: A program to demonstrate the use of For Each loop.

```

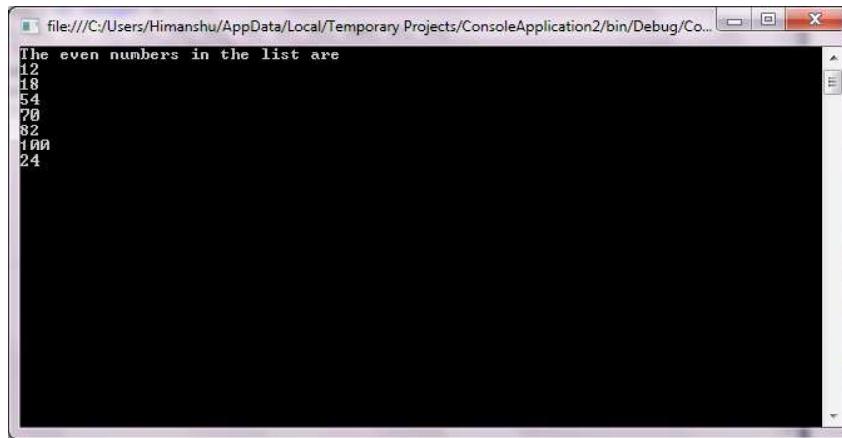
Imports System.Console
Module Module1
    Sub Main()
        Dim numberSeq() As Integer =
{12, 11, 21, 13, 15, 18, 54, 65, 70, 82, 99, 100, 24, 33}
        WriteLine("The even numbers in the list are")
        For Each number As Integer In numberSeq
            If number Mod 2 = 0 Then
                WriteLine(number.ToString)
                ' Display the number

```

```
End If  
Next  
Read()  
End Sub  
End Module
```

Control Statements

The output of the program is



NOTES

The While Loop

The `while` loop is used in those situations, where the number of iterations is not known in advance. Thus, unlike `For` loop, the `While` loop is non-deterministic in nature.

The syntax of the `While` loop is:

```
While condition  
statements  
End While
```

The following points should be noted about the `While` loop:

- Unlike `For` loops, the control variable must be declared and initialized before the `While` loop and needs to be updated within the body of the `While` loop.
- The `While` loop executes as long as condition evaluates to `True`. If condition evaluates to `False` in the first iteration, then the body of `While` loop never executes. Since it places the condition to be evaluated at the beginning of the loop, it is an **Entry controlled loop**.
- `While` loop can have more than one expression in its condition. However, such multiple expressions must be separated by commas and are executed in the order of their appearance.

Example 5.6: A program to demonstrate the use of `While` loop.

```
Imports System.Console  
Module Module1
```

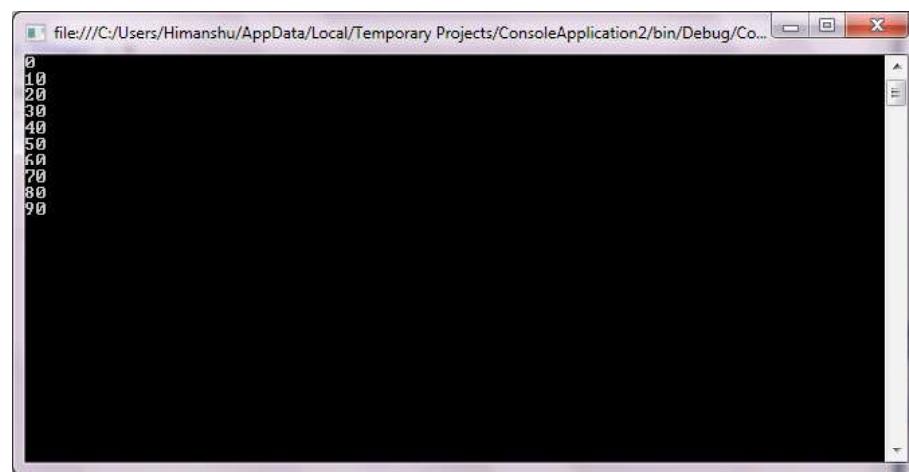
NOTES

```

Sub Main()
    Dim n As Integer = 0
    While n < 100
        WriteLine(n)
        n += 10
    End While
    Read()
End Sub
End Module

```

The output of the program is



The Do-Loop

As discussed earlier, in a `While` loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to `False`, the body of the loop is not executed even once. However, if the body of the loop is to be executed at least once, no matter whether the initial state of the condition is `True` or `False`, the Do-Loop is used. This loop places the condition to be evaluated at the end of the loop. Thus, it is an **Exit controlled loop**.

The syntax of the Do-Loop is:

```

do
statements
Loop {While|Until} condition

```

Example 5.7: A program to demonstrate the use of Do-Loop .

```

Imports System.Console
Module Module1
    Sub Main()
        Dim n As Integer
        Dim sum As Integer
        WriteLine("Enter the numbers (to stop enter 0)")

```

```

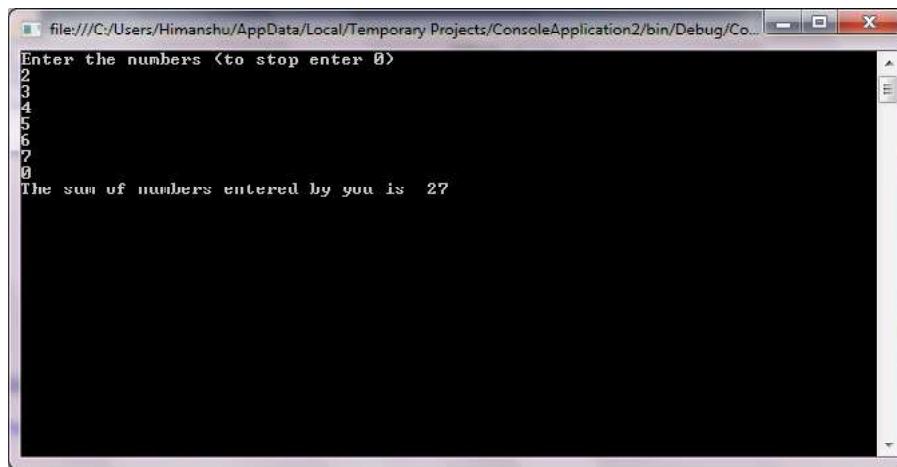
Do
n = Integer.Parse(ReadLine())
sum = sum + n
Loop While n <> 0
WriteLine("The sum of numbers entered by you is " &
Str(sum))
Read()
End Sub
End Module

```

Control Statements

NOTES

The output of the program is



If we replace the keyword `While` with `Until` in the Do-Loop, the statements inside the loop will be executed until the condition becomes True. That is, the loop will continue as long as the condition is False.

For example, if we replace the keyword `While` in Example 5.7 with the keyword `Until`, the code becomes as follows:

```

Imports System.Console
Module Module1
    Sub Main()
        Dim n As Integer
        Dim sum As Integer
        WriteLine("Enter the numbers (to stop enter 0)")
        Do
            n = Integer.Parse(ReadLine())
            sum = sum + n
        Loop Until n = 0 'Notice that the condition is changed
        WriteLine("The sum of numbers entered by you is " &
Str(sum))
        Read()
    End Sub
End Module

```

```

        End Sub
    End Module

```

The output of the program will be same.

NOTES

With Statement

With statement allows you to perform a series of statements on a specified object without qualifying the name of the object again and again. If you are using a With block, then you do not need to repetitively type the qualification path. It also reduces the risk of mistyping one of its elements. Note that With statement is not a loop but it is very useful and acts as a loop. The syntax of With statement is:

```

With object
statements
End With

```

Note that the term **object** used in the syntax of the With statement generally is of class or structure type. **statements** specifies the statements that are being executed and **End With** ends the definition of the With block.

Example 5.8: A program to demonstrate the use of With statement.

```

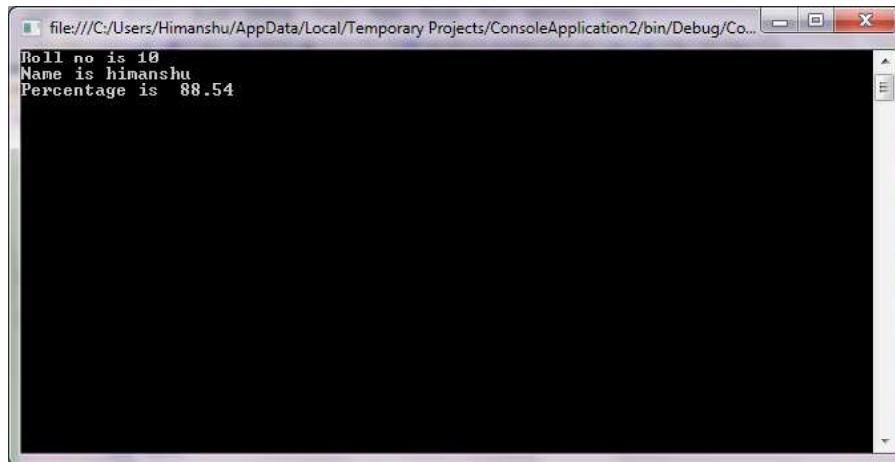
Imports System.Console
Structure student           'defining a structure student
    Public rollno As Integer      'members of the
    structure
    Public name As String
    Public percent As Single
End Structure
Module Module1
    Sub main()
        Dim s As student
        With s 'no need to qualify rollno, name and
        percent with s
            .rollno = 10
            .name = "himanshu"
            .percent = 88.54
        End With
        With s
            WriteLine("Roll no is" & Str(.rollno))
            WriteLine("Name is " & .name)
            WriteLine("Percentage is " & Str(.percent))
        End With
        Read()
    End Sub

```

```
End Sub  
End Module
```

Control Statements

The output of the program is



NOTES

Check Your Progress

1. What is a Select . . . Case statement in VB.NET? When is it used?
2. What are iteration statements? Name the various iteration statements used in VB.NET.

5.3 PROCEDURES IN VB.NET

A block of statements which are written to accomplish a single task, for example, adding two numbers, sorting a list of numbers, etc., is known as a **procedure**. The procedure is executed only when it is called from some other place in the program. The calling code could be a statement or an expression within a statement. In visual basic, there are two types of procedures:

- **Sub Procedures:** A Sub procedure, after running the code, does not return any value to the calling code. It is enclosed by Sub and End Sub statements.
- **Functions:** A function, after running the code, returns a value to the calling code. It is enclosed by Function and End Function statements.
Once the procedure execution is over, the control is returned back to the calling code. This can be done by using the Return, Exit, or End statement.
- **Return Statement:** It passes the control to the calling code immediately and the statements followed by the Return statement are not executed. A same procedure can have more than one Return statement.

NOTES

- **Exit Statement:** Just like the Return statement, Exit statement also returns the control to the calling code immediately. The statements followed by the Exit statement are not executed. A same procedure can have more than one Exit statement. In case of a Sub procedure, the Return statement is equivalent to Exit Sub statement; however, in case of a function, the Return statement returns a value back to the calling code, whereas Exit statement simply returns the control to the calling code.
- **End Statement:** The End statement also returns the control immediately to the calling code. However, unlike Return and Exit statements, which can be placed anywhere in the procedure, the End statement can be placed only at the end of the procedure. In addition, there can be only one End statement in a procedure.

Advantages of Procedures

As we have discussed, the procedures allow logical grouping of code into tasks. Dividing a complex application into procedures makes the code more flexible and easier to debug and maintain. Creating procedures has various other benefits, such as:

- After a procedure is created and tested; it can be called from different places in an application without writing the entire code again and again. Thus, the code in a procedure is reusable.
- The applications consisting of procedures are easier to debug and maintain, because creating procedures in an application makes it easier to trace the source of an error in a procedure as there is no need to test the entire application in order to look for errors.

5.3.1 Sub Procedures

Earlier, in this unit we have written programs containing only Main sub procedure. All the executable code is placed in the Main Sub procedure. The Main Sub procedure is called automatically when the program execution is started. In addition to Main, we can also create our own Sub procedures. This can be done as shown below:

```
Sub Main()
...   'body of main
End Sub
Sub hello()
Console.WriteLine("This is how a procedure is created")
End Sub
```

Calling the Sub Procedure

Merely, writing a Sub procedure is not enough to execute the statements written in it. The Sub procedure is executed only when it is called from some other procedure. For example, consider the following code segment.

```
Sub Main()
    hello()
End Sub
Sub hello()
    Console.WriteLine("This is how a procedure is created")
End Sub
```

In the above code segment, the Sub procedure `hello()` is called in the `Main` Sub procedure.

NOTES

Parameters and Arguments

A **parameter** represents a value that is supplied to the procedure when it is defined. The procedure's declaration defines its parameters. A procedure can be defined with no parameters, one parameter, or more than one. The part of the procedure definition that specifies the parameters is called the **parameter list**.

An **argument** represents the actual value that is supplied to a procedure parameter when the procedure is called. The calling code supplies the arguments when it calls the procedure. The part of the procedure call that specifies the arguments is called the **argument list**. The arguments which are passed to the procedure are enclosed in the parenthesis.

For example, suppose you want to display a string entered by the user instead of text "This is a how a procedure is created". For this, you need to pass a variable of type `String` in the parameter list of the Sub procedure. In addition, the actual string which is to be displayed is passed when the Sub procedure is called as shown in the following code.

```
Sub Main()
    hello("Passing actual values to the Sub procedure")

End Sub
Sub hello(ByVal strText As String)
    Console.WriteLine(strText)
End Sub
```

Passing Parameters using `ByVal` or `ByRef`

There are two ways of passing parameters to the Sub procedure: `ByVal` and `ByRef`. In `ByVal` method, a copy of a variable is passed to the Sub procedure. Any changes can be made to the copy and the original variable will not be

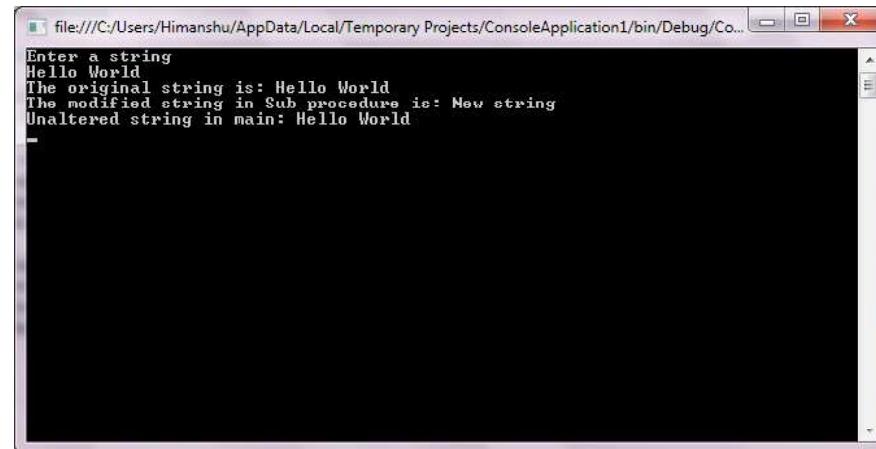
NOTES

altered. Whereas, in `ByRef` method, instead of passing a copy of the variable, a reference (or address) of the variable is sent. Thus, the changes are actually made to the original variable. For example, in the above code segment, `strText` is passed `ByVal` to the `hello()` Sub procedure.

Example 5.9: A program to demonstrate the concept of passing parameters `ByVal` in a Sub procedure.

```
Imports System.Console
Module Module1
    Sub Main()
        Dim Str1 As String
        WriteLine("Enter a string")
        Str1 = ReadLine()
        message1(Str1)
        WriteLine("Unaltered string in main: " & Str1)
        ReadLine()
    End Sub
    Sub message1(ByVal strText As String)
        WriteLine("The original string is: " & strText)
        strText = "New string"
        WriteLine("The modified string in Sub procedure
is: " & strText)
    End Sub
End Module
```

The output of the program is



It is clear from the output that the changes made to the variable `strText` in the Sub procedure are not reflected in the `Main` because the parameter is passed `ByVal`.

Example 5.10: A program to demonstrate the concept of passing parameters `ByRef` in a Sub procedure.

```

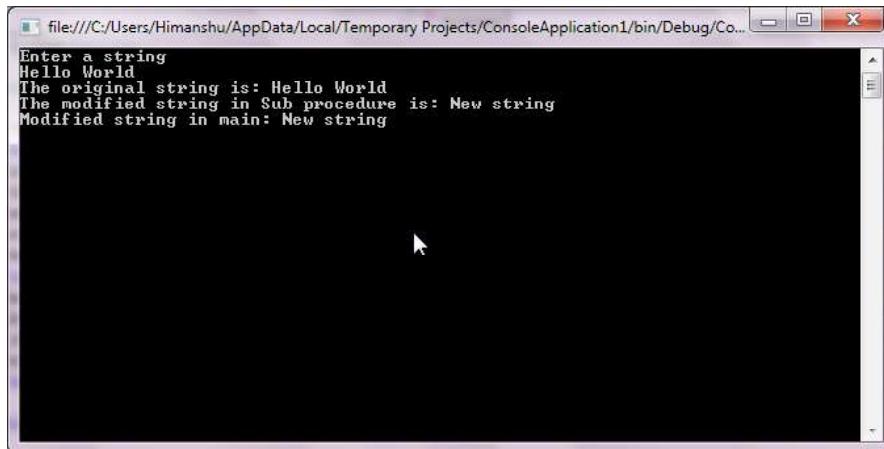
Imports System.Console
Module Module1
    Sub Main()
        Dim Str1 As String
        WriteLine("Enter a string")
        Str1 = ReadLine()
        message1(Str1)
        WriteLine("Modified string in main: " & Str1)
        ReadLine()
    End Sub
    Sub message1(ByRef strText As String)
        WriteLine("The original string is: " & strText)
        strText = "New string"
        WriteLine("The modified string in Sub procedure
is: " & strText)
    End Sub
End Module

```

Control Statements

NOTES

The output of the program is



5.3.2 Functions

Like Sub procedures, functions are also well-defined named group of statements that are aimed at accomplishing a specific task or action in the program. They are used to combine particular set of instructions that needs to be accessed repeatedly in a program. The use of functions prevents the repetition of code and reduces the size of the entire program. This is because a function's code is written once and can be used at different places in the program by calling the function. Like variables, functions also need to be declared before they are used in programs. A function can receive arguments and can even return a value.

A function is declared in the same way as Sub procedure is declared. The only difference is that a function is enclosed within Function and End

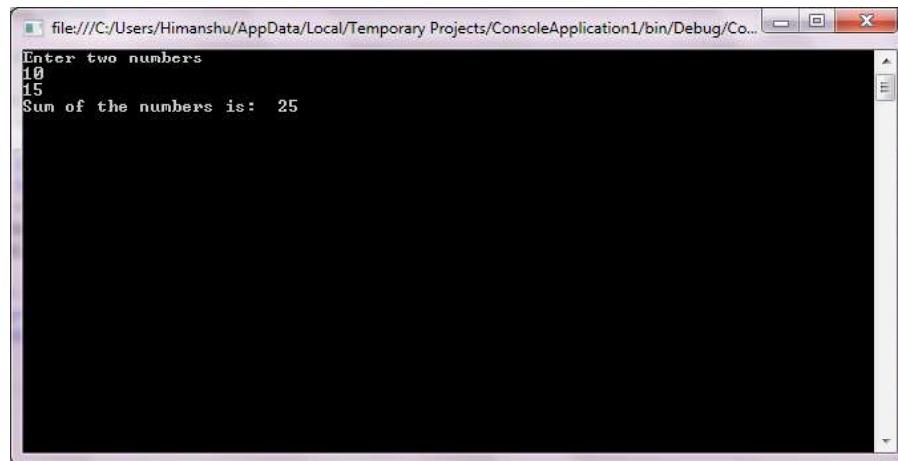
Function statements, and a return type also needs to be specified that indicates the data types of the value returned by the function.

Example 5.11: A program to demonstrate how to create a function returning a value.

NOTES

```
Imports System.Console  
Module Module1  
    Sub Main()  
        Dim A, B, sum As Integer  
        WriteLine("Enter two numbers")  
        A = Integer.Parse(ReadLine())  
        B = Integer.Parse(ReadLine())  
        sum = Addition(A, B)  
        WriteLine("Sum of the numbers is: " & Str(sum))  
        ReadLine()  
    End Sub  
    Function Addition(ByVal n1 As Integer, ByVal n2 As Integer) As Integer  
        Return n1 + n2  
    End Function  
End Module
```

The output of the program is



5.3.3 MsgBox() and InputBox() Functions

Functions are classified into two types: *built-in functions* (or *internal functions*) and *user-defined functions*. Example 5.11 demonstrates the creation of a user-defined function. The two commonly used built-in functions are `MsgBox()` and `InputBox()` which are used for input/output operations.

MsgBox () Function

`MsgBox()` function is used to display messages in the form of message box instead of on the console output. It prompts the user to click the button displayed on the message box. The syntax for using `MsgBox()` function is:

```
displayMsg=Msgbox(Prompt, Style_Value, Title)
where,
```

`Prompt` refers to the message that is displayed in the message box. Its maximum length is 1024 characters.

`Style_Value` specifies the type of command button that is displayed on the message box.

`Title` refers to the text that is displayed in the title bar of the message box.

The parameter `Style_Value` can take various values, which are listed in Table 5.1.

NOTES

Table 5.1 Style Values in `MsgBox()`

Style_Value	Constant	Button(s)/Message Displayed
0	<code>vbOKOnly</code>	OK button
1	<code>vbOKCancel</code>	OK and Cancel buttons
2	<code>vbAbortRetryIgnore</code>	Abort, Retry, and Ignore buttons
3	<code>vbYesNoCancel</code>	Yes, No, and Cancel buttons
4	<code>vbYesNo</code>	Yes and No buttons
5	<code>vbRetryCancel</code>	Retry and Cancel buttons
16	<code>vbCritical</code>	Displays Critical message
32	<code>vbQuestion</code>	Displays Warning query icon
48	<code>vbExclamation</code>	Displays Warning message icon
64	<code>vbInformation</code>	Displays Information message icon
0	<code>vbDefaultButton1</code>	Makes the first button as default button
256	<code>vbDefaultButton2</code>	Makes the second button as default button
512	<code>vbDefaultButton3</code>	Makes the third button as default button
0	<code>vbApplicationModal</code>	Shows the application modal message box—the user cannot work in the current application without responding to the message box
4096	<code>vbSystemModal</code>	Shows the system modal message box—all the applications are unavailable until the message box is dismissed
65536	<code>vbMsgBoxSetForeground</code>	Displays the message box window as the foreground window
524288	<code>vbMsgBoxRight</code>	Text displayed in the message box will be right-aligned
1048576	<code>vbMsgBoxRtlReading</code>	Text displayed in the message box will appear as right-to-left—used if message to displayed is in other languages such as Arabic and Hebrew

To make the program more readable, the integer values in the second argument of the `MsgBox()` function can be replaced with the named constant. For example, the following statement:

NOTES

```
displayMsg=Msgbox("Click cancel to skip", 1, message1)
```

can also be written as

```
displayMsg=Msgbox("Click cancel to skip", vbOkCancel,
message1)
```

The variable `displayMsg` will store the value returned by the `MsgBox()` function, when the user clicks a button on the message box. The different values returned by the `MsgBox()` are listed in Table 5.2.

Table 5.2 Values Returned by `MsgBox()` Function

Button Clicked	Value
OK button	1
Cancel buttons	2
Abort button	3
Retry buttons	4
Ignore buttons	5
Yes button	6
No button	7

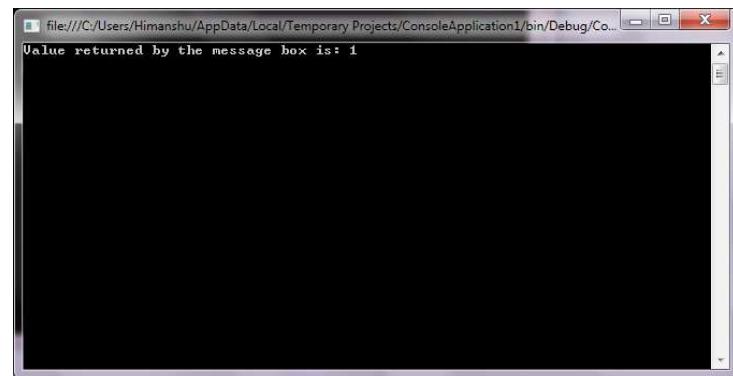
Example 5.12: A program to demonstrate the use of `MsgBox()` function.

```
Imports System.Console
Module Module1
    Sub Main()
        Dim displayMsg As Integer
        displayMsg = MsgBox("Hello", 0, "Message Box")
        WriteLine("Value returned by the message box is:"
& Str(displayMsg))
        ReadLine()
    End Sub
End Module
```

The output of the program is



When the user clicks the **OK** button, the following message will be displayed.



NOTES

InputBox() Function

`InputBox()` function prompts the user to enter a value or message in the dialog box. The syntax of the `InputBox()` function is:

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

where,

`displayMsg` is generally of `String` type that accepts the message entered by the user.

`Prompt` refers to the text which is displayed in the `InputBox`. This argument is mandatory.

`Title` refers to the text that is displayed in the title bar of the `InputBox`.

`Default-text` is an optional argument. It specifies the string that appears in the `TextBox` by default, when the `InputBox` is displayed to the user. The user can overwrite this text by providing his/her own text.

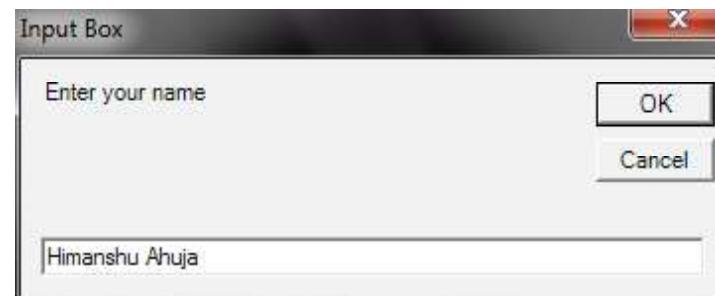
`X-position, Y-position` specifies the coordinates or position of the `InputBox` on the screen.

Example 5.13: A program to demonstrate the use of `InputBox()`

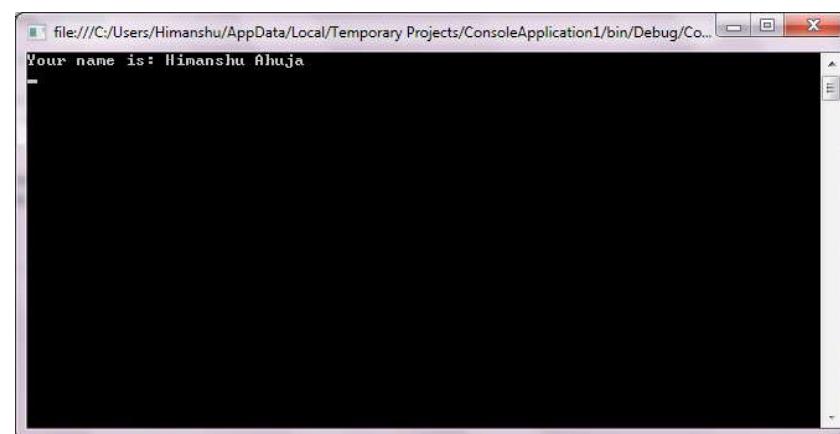
```
Imports System.Console
Module Module1
    Sub Main()
        Dim name As String
        name = InputBox("Enter your name", "Input Box",
        "Default text",
        200, 200)
        WriteLine("Your name is: " & name)
        ReadLine()
    End Sub
End Module
```

NOTES

The output of the program is



When the user enters his or her name, and clicks **OK**, the following message is displayed. If the user clicks the **Cancel** button, no message will be displayed.



Check Your Progress

3. What is a procedure?
4. What are the two types of procedures in VB.NET?

5.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Select... Case in VB.NET is also a kind of selection statement. It is used in those situations where there are many choices and for those choices, more than one condition or case is required. The Select Case statement makes use of a variable or an expression as a test variable. The flow of the program is controlled by this variable therefore, this variable is also known as control variable. Based on this variable an expression from the expression list is selected.

2. The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as iteration statements. That is, as long as the condition evaluates to True, the set of statement(s) is executed. The various iteration statements used in VB.NET are as follows:

- For statement
- For Each statement
- While Loop
- Do-Loop

3. A block of statements which are written to accomplish a single task, for example, adding two numbers, sorting a list of numbers, etc., is known as a procedure. The procedure is executed only when it is called from some other place in the program. The calling code could be a statement or an expression within a statement.

4. In visual basic, there are two types of procedures:

- **Sub procedures:** A Sub procedure, after running the code, does not return any value to the calling code. It is enclosed by Sub and End Sub statements.
- **Functions:** A function, after running the code, returns a value to the calling code. It is enclosed by Function and End Function statements.

NOTES

5.5 SUMMARY

- In VB.NET, the control statements are broadly classified into three categories, namely, selection statements, iteration statements and jump statements. All these control statements are commonly used with the logical tests or test conditions to alter the flow of control conditionally or unconditionally.
- Selection statements, also known as conditional statements, are used to make decisions based on a given condition. If the condition evaluates to True, a set of statements is executed, otherwise another set of statements is executed.
- Various types of selection statements supported by VB.NET are If statement, If-Else statement, Nested If-Else statement, and Select...Case statement.
- The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are

NOTES

known as iteration statements. That is, as long as the condition evaluates to True, the set of statement(s) is executed.

- The various iteration statements used in VB.NET are as follows For statement, For Each statement, While loop, and Do-Loop.
- A block of statements which are written to accomplish a single task, for example, adding two numbers, sorting a list of numbers, etc., is known as a procedure. The procedure is executed only when it is called from some other place in the program. The calling code could be a statement or an expression within a statement. In visual basic, there are two types of procedures namely Sub procedure and function.
- Like Sub procedures, functions are also well-defined named group of statements that are aimed at accomplishing a specific task or action in the program. They are used to combine particular set of instructions that needs to be accessed repeatedly in a program.
- Functions are classified into two types: built-in functions (or internal functions) and user-defined functions.
- The two commonly used built-in functions are MsgBox() and InputBox() which are used for input/output operations.
- The term exception is an abbreviation for the phrase “exceptional event”. It is an unpredicted event that occurs while the program is executing and thus, disrupts the normal flow of the program or terminates the program abnormally.

5.6 KEY WORDS

- **Selection Statements (or Conditional Statements):** The statements which are used to make decisions based on a given condition and if the condition evaluates to True, a set of statements is executed, otherwise another set of statements is executed.
- **Iteration Statements:** The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied.
- **Procedure:** A block of statements which are written to accomplish a single task.
- **Sub Procedures:** A kind of procedure that does not return any value to the calling code and it is enclosed by Sub and End Sub statements.
- **Functions:** A kind of procedure that returns a value to the calling code enclosed by Function and End Function statements.

5.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. How is a Sub procedure different from a function?
2. What are the benefits of creating procedures?
3. Write the working of Try...Catch...Finally block with the help of a diagram.
4. Write a program to find the area of a circle.
5. Write a program to check whether a given number is odd or even.

NOTES**Long Answer Questions**

1. What are the two ways of passing parameters to a procedure? Explain with the help of suitable examples.
2. Write a program to demonstrate the use of Try...Catch...Finally statements in VB.NET.
3. Explain the MsgBox () and InputBox () functions. Give syntax for both.
4. Write a menu-driven program that performs basic arithmetic operations (+, -, *, /, \, and mod) depending on the user's choice. Make use of select case statement.

5.8 FURTHER READINGS

Holzner, Steven. 2002. *Visual Basic .NET Black Book*, United Kingdoms: Coriolis Group Books.

Deitel, Harvey M. 2001. *Visual Basic.NET How to Program*, 2nd edition. New Jersey: Prentice Hall.

UNIT 6 WINDOWS FORMS

NOTES**Structure**

- 6.0 Introduction
 - 6.1 Objectives
 - 6.2 Windows Forms
 - 6.2.1 MDI Forms
 - 6.3 Event
 - 6.4 MSGBOX
 - 6.4.1 Simple MessageBox
 - 6.4.2 MessageBox with Title
 - 6.4.3 MessageBox with Buttons
 - 6.4.4 MessageBox with Icon
 - 6.4.5 MessageBox with Default Button
 - 6.4.6 MessageBox with Message Options
 - 6.4.7 MessageBox with Help Button
 - 6.5 InputBox()
 - 6.6 DialogBox
 - 6.7 RichTextBox
 - 6.8 Label Class
 - 6.9 LinkLabel Control
 - 6.10 Passing Forms
 - 6.11 Answers to Check Your Progress Questions
 - 6.12 Summary
 - 6.13 Key Words
 - 6.14 Self Assessment Questions and Exercises
 - 6.15 Further Readings
-

6.0 INTRODUCTION

Windows Forms is a Graphical User Interface (GUI) class library included in a .Net Framework. The primary purpose of providing this is to make the development of applications for desktop, tablet or PCs simpler and easier. It is also known as WinForms and the applications created using this (Windows Forms or WinForms) are known as the Windows Forms Applications. These applications run on the desktops. WinForms is suitable for developing only the Windows Forms Applications (or the desktop application) and cannot be used to develop web applications. WinForms applications can contain various types of controls like list boxes, names, tooltip, labels etc.

6.1 OBJECTIVES

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

- Understand and create the Windows form applications
- Implement a MDI form

- Discuss the concept of events and its types
- Create and add various controls like MessageBox, DialogBox, RichTextBox control to a form

Windows Forms

6.2 WINDOWS FORMS

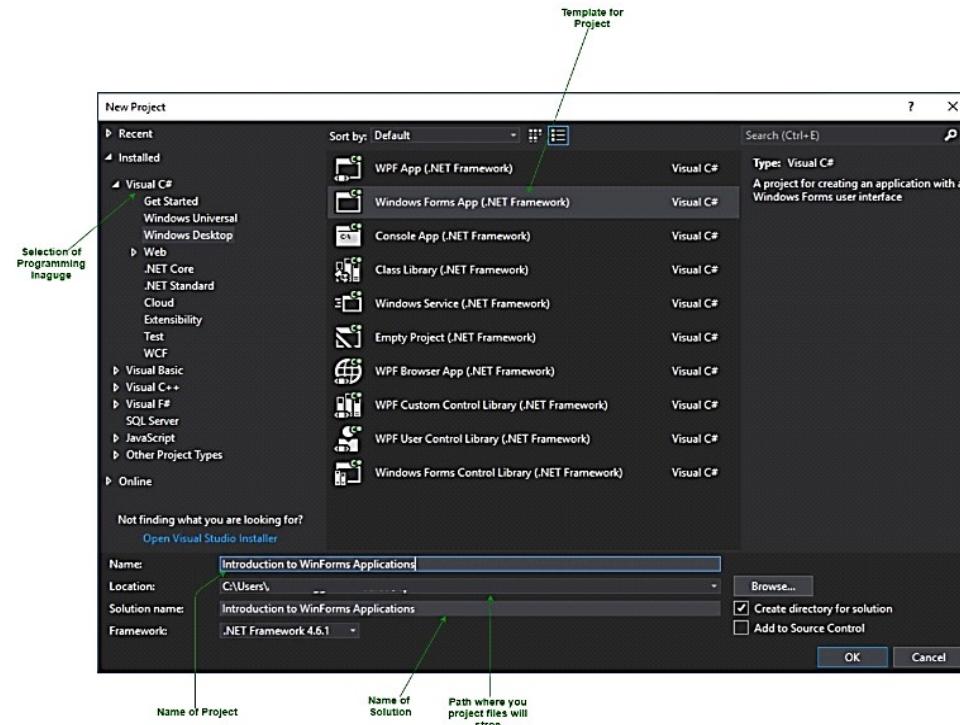
NOTES

Windows Forms (WinForms) is a graphical (GUI) class library included as a part of Microsoft.NET Framework or Mono Framework, giving a common platform to compose rich customer applications for workstation, work area, and tablet PCs. It is viewed as a trade for the prior and increasingly complex C++ based Microsoft Foundation Class Library. It doesn't offer a practically identical paradigm and used as a platform for the UI level in a multi-level solution.

Steps for creating a Windows Forms Application using Visual Studio 2017

Following are the steps for creating a Windows form application.

1. Open the Visual Studio, then Go to File -> New -> Project to create a new project and then select the language as Visual C# from the left menu. You can see on Windows Forms App (.NET Framework) in the midst of current window. After that give the name of the project and Click OK. The following window will appear.



NOTES

It acts as a containers that contains the projects and files which might be required by the programmer at any point of time.

- Once the WinForms is open, a window will be displayed that is divided into three main parts as discussed below:

- Main Window or Editor Window:** This is the window where designing of form can be done and code can be written. Initially the form has a blank layout and double click on the form will take you to the code section.
- Solution Explorer Window:** This provides a navigation between different items/ objects present in the form. It displays all the different forms present in the project and when any particular form (or file) is selected, then particular information related to that form will be displayed in the property window.
- Properties Window:** This window allows user to change the various properties associated with the selected item/object in the solution explorer. This window will also allow users to change the properties of different controls or components (eg. TextBox, label, button etc.) which are added to the forms.

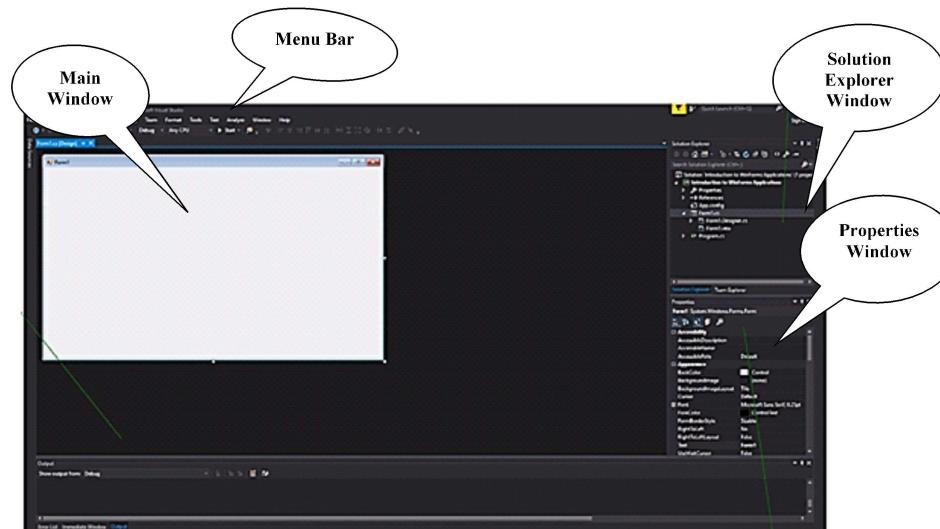
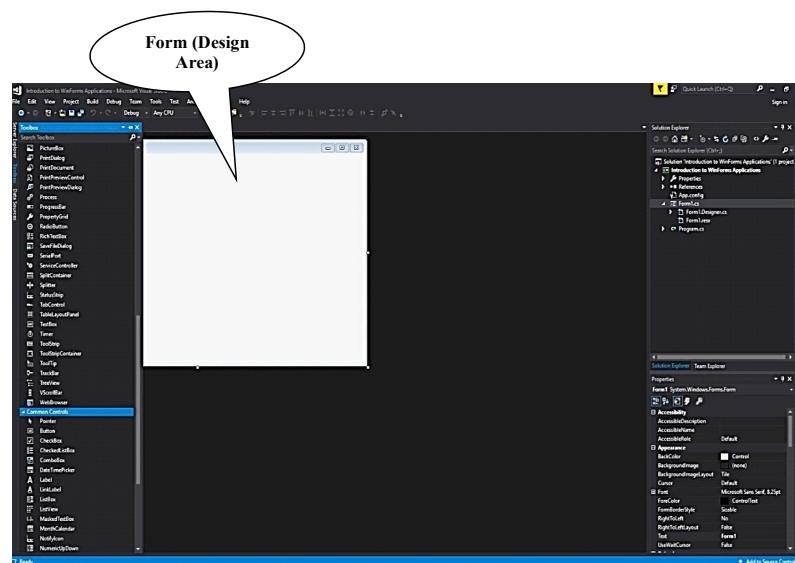


Fig. 6.1 Window Form

The layout of the window can also be set to default by clicking **Window -> Reset Window Layout** in Visual Studio Menu.

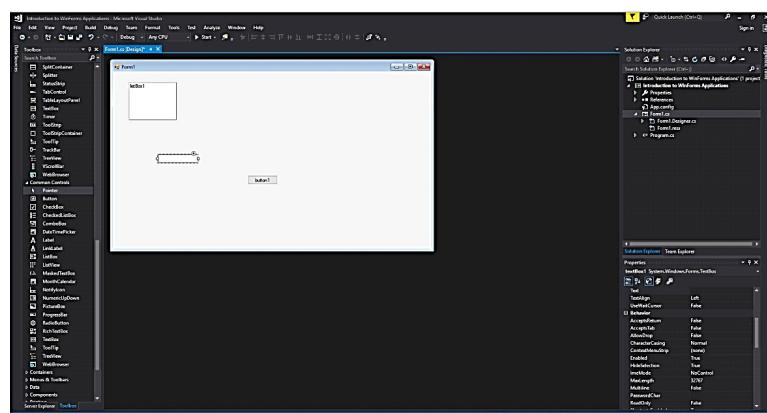
Steps for adding various controls to the WinForms application

- In order to add various controls directly to your WinForms application locate the Toolbox tab at the left side of the Visual Studio where, a list of controls is displayed. The most frequently used controls are present under the Common Controls option in the Toolbox tab.

**NOTES****Fig. 6.2 Form**

2. In order to bring the control on the created form, just drag and drop the required control on to the form. Below is the screenshot displaying the controls, listBox, TextBox and a button. The properties of the control can be accessed and changed by selecting the particular control. The properties associated with the control will be displayed in the properties window on the right side, when the control is selected.

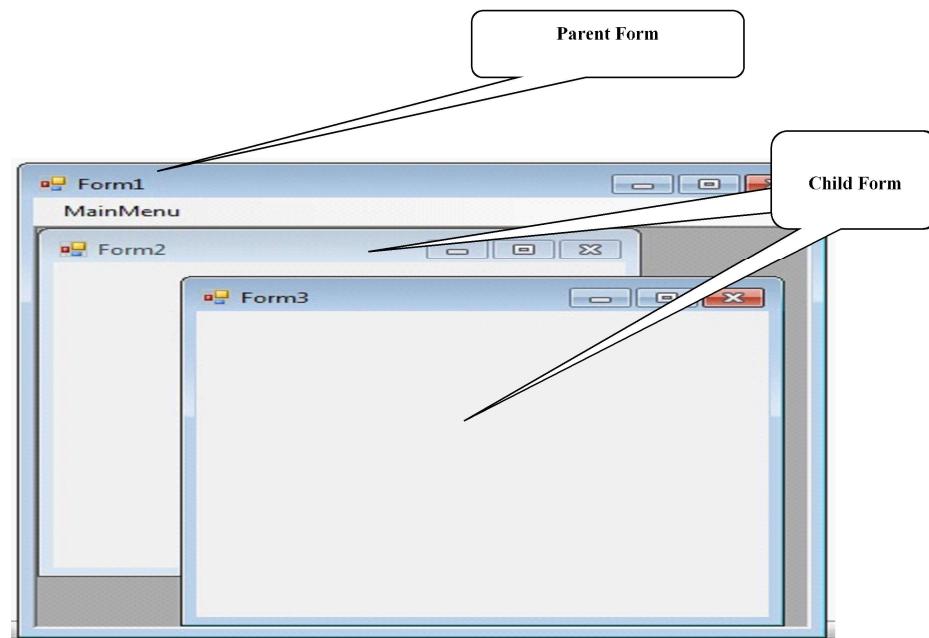
For example, in the screenshot below, the selected control is a TextBox and its associated properties such as TextAlign, MaxLength, MultiLine etc. are shown in the Properties Window. As per the need of the applications, the values of these properties can be changed. The code for that control and for a particular property is automatically added in the background.



3. The program can be run using F5 key or Play button present in the toolbar of Visual Studio. Alternatively, Debug->Start Debugging menu in the menu bar can also be used to stop the program. Similarly, to stop the program, pause button can be used present in the ToolBar.

NOTES**Fig. 6.3 Form 1****6.2.1 MDI Forms**

Multiple Document Interface or MDI Application provides a programming interface to create applications that enables to display multiple document at the same time, while displaying each documents in its own window. In other words, many source files and design views can be open at once. Each file or document has its own separate space with its own controls for scrolling. The user can simply navigate between the documents by simply moving the cursor from one space to another. However, the view space in MDI applications is confined to the application's window or client area. Figure 6.4 shows each document (Form 1, Form2, Form3) have been displayed in the separate child window within the client area. This is in contrast with Single Document Interface (SDI), where single document can only be accessed and manipulated at a time. Notepad, paint are some examples of SDI.

**Fig. 6.4 Multiple-Document Interfaces (MDI)**

MDI Parent Form

Windows Forms

MDI applications consists of two types of forms, main form (or parent form) and the child form. The parent form does not itself display any data, whereas the child form is used to display documents and appears only within the parent form. The parent form coordinates all the child forms or reports that are open at present. The main form is called as the *MDI parent* and the child forms are called as *MDI children*. In a way, MDI parent acts as a background for the application and contains MDI child forms. You may have seen such options in various Windows applications under a Windows menu, for instance, Cascade, Tile Vertical, and so forth.

NOTES

Steps for creating a MDI parent form with a menu bar

1. Create a Windows Application project.
2. In the **Properties** window, set the **IsMdiContainer** property to **true**.

This designates the form as an MDI container for child windows.

3. From the **Toolbox**, drag a **MenuStrip** control to the form. Create a top-level menu item with the **Text** property set to **&File** with submenu items called **&New** and **&Close**. Also create a top-level menu item called **&Window**.

The first menu will open and close menu items at run time, and the second menu will keep track of the open MDI child windows. At this point, you have created an MDI parent window.

4. Press **F5** to run the application.

MDI Child Form

MDI child forms are the important element of MDI application as the contents are displayed in these forms.

Steps for creating MDI child forms

1. Create a new Windows Forms project. In the **Properties Windows** for the form, set its **IsMdiContainer** property to **true**, and its **WindowState** property to **Maximized**.
2. This designates the form as an MDI container for child windows.
3. From the Toolbox, drag a **MenuStrip** control to the form. Set its **Text** property to **File**.
4. Click the ellipses (...) next to the **Items** property, and click **Add** to add two child tool strip menu items. Set the **Text** property for these items to **New** and **Window**.
5. In **Solution Explorer**, right-click the project, point to **Add**, and then select **Add New Item**.

NOTES

6. In the **Add New Item** dialog box, select **Windows Form** (in Visual Basic or in Visual C#) or **Windows Forms Application (.NET)** (in Visual C++) from the **Templates** pane. In the **Name** box, name the form **Form2**. Click the **Open** button to add the form to the project.

The MDI child form you created in this step is a standard Windows Form. As such, it has an **Opacity** property, which enables you to control the transparency of the form. However, the **Opacity** property was designed for top-level windows. Do not use it with MDI child forms, as painting problems can occur.

7. This form will be the template for your MDI child forms.
8. The **Windows Forms Designer** opens, displaying **Form2**.
9. From the **Toolbox**, drag a **RichTextBox** control to the form.
10. In the **Properties** window, set the **Anchor** property to **Top, Left** and the **Dock** property to **Fill**.
11. This causes the **RichTextBox** control to completely fill the area of the MDI child form, even when the form is resized.
12. Double click the **New** menu item to create a **Click** event handler for it.
13. Insert code similar to the following to create a new MDI child form when the user clicks the **New** menu item.

In the following code, the event handler handles the **Click** event for **MenuItem2**. Be aware that, depending on the specifics of your application architecture, your **New** menu item may not be **MenuItem2**.

C#

```
protected void MDIChildNew_Click(object sender,
System.EventArgs e)
{
    Form2 newMDIChild = new Form2();
    // Set the Parent Form of the Child window.
    newMDIChild.MdiParent = this;
    // Display the new form.
    newMDIChild.Show();
}
```

- In the drop-down list at the top of the **Properties** window, select the menu strip that corresponds to the **File** menu strip and set the **MdiWindowListItem** property to the **Window ToolStripMenuItem**.
- This will enable the **Window** menu to maintain a list of open MDI child windows with a check mark next to the active child window.
- Press F5 to run the application. By selecting **New** from the **File** menu, you can create new MDI child forms, which are kept track of in the **Window** menu item.

6.3 EVENT

A windows application is an *event-driven* application where an event is a message sent by an object to inform an application about the occurrence of an action also known as event. It is a way through which user can interact with different element/controls present in the form within the user interface, in any order they choose. It may be caused by clicking a mouse (MouseClick event) or a button (ButtonClick event). There are two terms related to an event as given below:

1. **Event Sender:** It is the object that sent the event.
2. **Event Receiver:** The object that receives the event sent by the event sender and responds accordingly.
3. **Event Handler:** A sub routine or procedure which defines the corrective action at the time of event creation.

NOTES

VB.Net is an event driven language and has mainly two types of events:

1. Mouse events
2. Keyboard events

Handling Mouse Events

When there is a movement in forms and controls, a mouse event occurs. The various mouse events related with a Control class has been described below:

- **MouseDown** – It occurs when a mouse button is pressed.
- **MouseEnter** – It occurs when the mouse pointer enters the control.
- **MouseHover** – It occurs when the mouse pointer hovers over the control.
- **MouseLeave** – It occurs when the mouse pointer leaves the control.
- **MouseMove** – It occurs when the mouse pointer moves with the control.
- **MouseUp** – It occurs when the mouse pointer is over the control and the mouse button is released.
- **MouseWheel** – It occurs when the mouse wheel moves and the control has focus.

To handle the mouse events, events handlers of mouse events get an argument of type MouseEventArgs, which has the following properties:

- **Buttons** – It indicates the mouse button is pressed.
- **Clicks** – It indicates the number of clicks.
- **Delta** – It indicates the number of detents the mouse wheel rotated.
- **X** – It indicates the x-coordinate of mouse click.
- **Y** – It indicates the y-coordinate of mouse click.

NOTES**Handling Keyboard Events**

The various keyboards events related with a Control class are as follows:

- **KeyDown** – It occurs when a key is pressed down and the control has focus.
- **KeyPress** –It occurs when a key is pressed and the control has focus.
- **KeyUp** – It occurs when a key is released while the control has focus.

Events handlers of keyboard events also get an argument of type **KeyboardEventArgs**, which has the following properties:

- **Alt** – It indicates whether the ALT key is pressed.
- **Control** – It indicates whether the CTRL key is pressed.
- **Handled** – It indicates whether the event is handled.
- **KeyCode** – It stores the keyboard code for the event.
- **KeyData** – It stores the keyboard data for the event.
- **KeyValue** – It stores the keyboard value for the event.
- **Modifiers** – It indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed.
- **Shift** – It indicates if the Shift key is pressed.

The arguments of type **KeyEventEventArgs** for event handlers of KeyUp and KeyDown events has the following properties:

- **Handled** – It indicates if the KeyPress event is handled.
- **KeyChar** – It stores the character corresponding to the key pressed.

Check Your Progress

1. What are WinForms?
2. What are the three main parts of WinForm window?
3. What is the use of MDI application?
4. What are the two types of events?

6.4 MSGBOX

A message box is a type of dialog box that is used to communicate any piece of information with the user. MessageBox control in Windows Forms is used to display a message with the given text and action buttons. You can also use MessageBox control to add additional options such as a caption, an icon, or help buttons. There is no provision to type anything by the user. It just briefs the user about the instructions with text, buttons and symbols to proceed further.

MessageBox class is generally used for general message display and user response.

Windows Forms

The Show method of MessageBox is used to display a user specific message in a dialog box and waits for the user to click a button.

NOTES

6.4.1 Simple MessageBox

The simplest form of a MessageBox is a dialog with a text and OK button. When you click OK button, the box disappears.

The general form is:

```
MessageBox.Show(string message, string caption,  
MessageBoxButtons b, MessageBoxIcon ic)
```

The following code creates a simple MessageBox.

```
string message = "Welcome to Microsoft Visual Basic";  
MessageBox.Show(message);
```

The output of above code is:

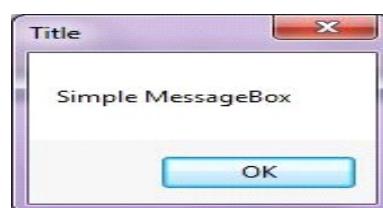


6.4.2 MessageBox with Title

The following code snippet creates a simple MessageBox with a title.

```
string message = "Simple MessageBox";  
string title = "Title";  
MessageBox.Show(message, title);
```

The output of the code will be:



6.4.3 MessageBox with Buttons

A number of buttons combination can be created on a MessageBox such as combination of Yes/No, OK/Cancel etc. The MessageBoxButtons enumeration represents the buttons to be displayed on a MessageBox and has following values.

- OK
- OK/Cancel

NOTES

- Abort/Retry/Ignore
- Yes/No/Cancel
- Yes/No
- Retry/Cancel

Let's take an example that creates a message box with its title and YES/NO buttons. Generally we call this message box at the time of closing an application. If we click on Yes button the application will be closed. If we click on NO button message box will disappear from the screen. The Show method returns a DialogResult enumeration.

```
string message = "Do you want to close this window?";
string title = "Close Window";
MessageBoxButtons buttons = MessageBoxButtons.YesNo;
DialogResult result = MessageBox.Show(message, title,
buttons);
if (result == DialogResult.Yes) {
this.Close();
} else {
// Do something
}
```

The output will be as shown below.



6.4.4 MessageBox with Icon

The dialog of a MessageBox can also have an icon. A MessageBoxIcon enumeration represents an icon to be displayed on a MessageBox. It has following values.

- None
- Hand
- Question
- Exclamation
- Asterisk
- Stop
- Error

- Warning
- Information

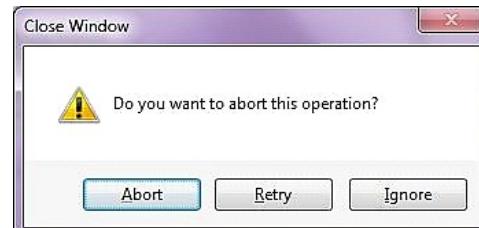
Windows Forms

Code for a MessageBox with a title, buttons, and an icon is given below.

```
string message = "Do you want to abort this operation?";
string title = "Close Window";
MessageBoxButtons buttons = MessageBoxButtons.AbortRetryIgnore;
DialogResult result = MessageBox.Show(message, title,
buttons, MessageBoxIcon.Warning);
if (result == DialogResult.Abort) {
this.Close();
}
elseif(result == DialogResult.Retry) {
    // Do nothing
}
else {
    // Do something
}
```

NOTES

Output:



6.4.5 MessageBox with Default Button

You can also set the default button on a MessageBox as per the requirements. When we will not set the default button in that case the first button will be the default button. The `MessageBoxDefaultButton` enumeration is used for this purpose and it has the following three values.

- Button1
- Button2
- Button3

Code for a MessageBox with a title, buttons, and an icon and setting the second button as a default button is given below.

```
string message = "Do you want to abort this operation?";
string title = "Close Window";
MessageBoxButtons buttons = MessageBoxButtons.AbortRetryIgnore;
```

NOTES

```

    DialogResult result = MessageBox.Show(message, title,
    buttons, MessageBoxIcon.Warning, MessageBoxDefault
    Button.Button2);
    if (result == DialogResult.Abort) {
        this.Close();
    }
    elseif(result == DialogResult.Retry) {
        // Do nothing
    }
    else {
        // Do something
    }

```

Output:**6.4.6 MessageBox with Message Options**

A Message box can also have a number of message options on it. `MessageBoxOptions` enumeration represents various options and has the following values.

- `ServiceNotification`
- `DefaultDesktopOnly`
- `RightAlign`
- `RtlReading`

Code for creating a `MessageBox` with various options is as follows.

```

    DialogResult result = MessageBox.Show(message, title,
    buttons, MessageBoxIcon.Warning, MessageBoxDefaultButton.
    Button2,
    MessageBoxButtons.RightAlign | MessageBoxButtons.
    RtlReading);

```

Output:

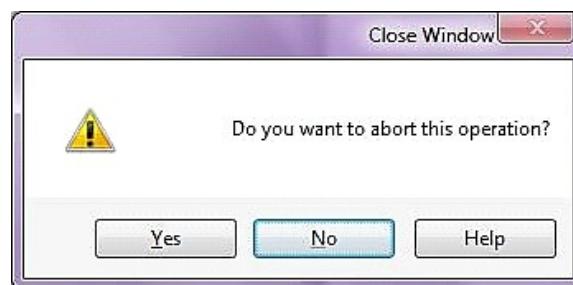
6.4.7 MessageBox with Help Button

For any type of help regarding the message box or any contents on this message box, you can create the help button on it.

Code for creating a MessageBox with a Help button is as follows.

```
DialogResult result = MessageBox.Show(message, title,
buttons, MessageBoxIcon.Warning, MessageBoxButtons.Default
Button.Button2, MessageBoxOptions.RightAlign, true);
```

Output:

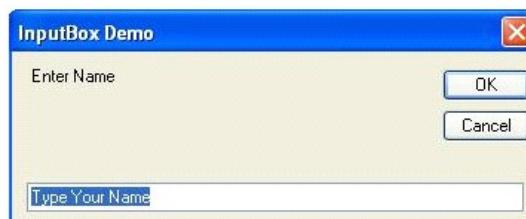


NOTES

6.5 INPUTBOX ()

An InputBox function displays a dialog box that prompts user to insert a value. The dialog box contains a text box for entering the value, OK and Cancel buttons and (optionally) a help button.

The sample input box is shown below:



Here the dialog box waits for either user input or click any button. When Ok button is pressed, the content of the TextBox is returned in the form of a string else a blank value is returned.

The general form of the function is given below:

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

where,

- **myMessage** is a variable that will hold the value of the input in the TextBox.
- **Prompt** is a compulsory argument as a string that specifies the message to be displayed in the InputBox dialog box.

NOTES

- **Title** is an optional argument as string that is displayed in the title bar of the `InputBox`. By default, the name of the application is displayed in the title bar if this argument is missing.

- **Default_text** is an optional argument as a string that specifies the default value in the `TextBox` of the `InputBox`. If this argument is missing from the `InputBox` function, then `TextBox` is displayed empty.
- **X-position** is an optional argument that specifies the distance in pixels of the `InputBox` from the left edge of the screen.
- **Y-position** is an optional argument that specifies the distance in pixels of the `InputBox` from the top edge of the screen.
- If both x-position and y-position arguments are missing then by default, the `InputBox` will be displayed at the center of the screen.

The configuration won't work in Visual Basic 2012 in light of the fact that `InputBox` is considered a namespace. Thus, you have to enter in the full reference to the `InputBox` namespace, which is `Microsoft.VisualBasic.InputBox(Prompt, Title, default_text, x-position, y-position)`.

Consider the following example.

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
    Dim userMsg As String
    userMsg = Microsoft.VisualBasic.InputBox("What is
your message?", "Message Entry Form", "Enter your messge
here", 500, 700)
    If userMsg <> "" Then
        MessageBox.Show(userMsg)
    Else
        MessageBox.Show("No Message")
    End If
End Sub"
```

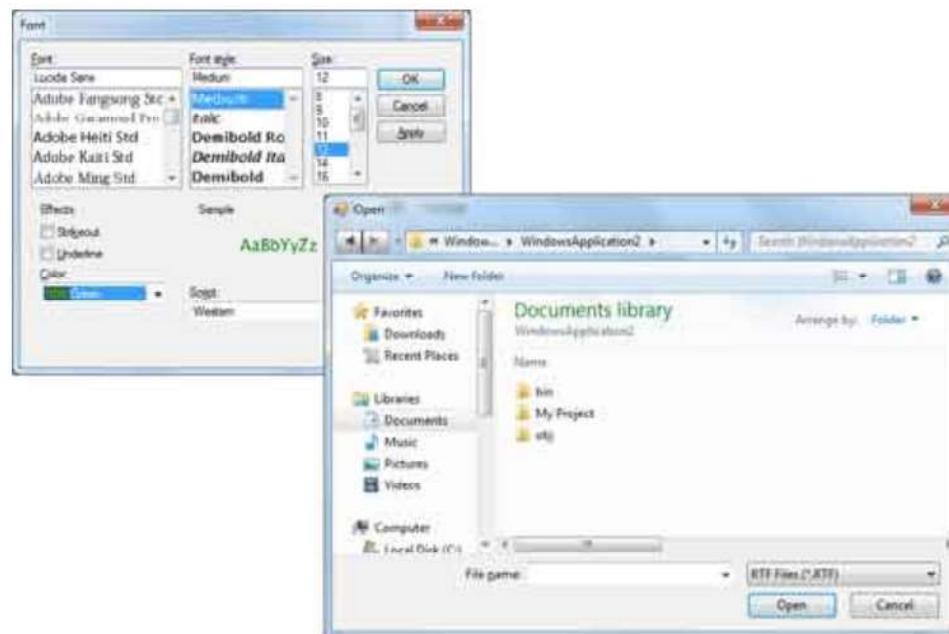
The `InputBox` will appear as shown below, when you press the command button.



6.6 DIALOGBOX

DialogBox is a way to interact with users and retrieve information. You must have come across with many built-in dialog boxes in windows for various tasks such as open and save a file, print a page, font name and size, providing colors, fonts, page setups etc. DialogBox is a way of getting information from the user and to show exactly how things are configured. In fact, for common operations, standard dialog boxes are used by all windows applications. Examples of some common dialog boxes are shown below in the screenshot. Common dialog boxes are implemented as any other controls in the toolbox and can be used in the interface. To use any DialogBox, drag the appropriate control from the DialogBox section and drop it on the form. The control can be activated from within the code the code by calling ShowDialog method. Generally these controls are not placed on the forms and are invisible at the runtime. They are displayed generally as and when needed as they are implemented as modal dialog boxes.

NOTES

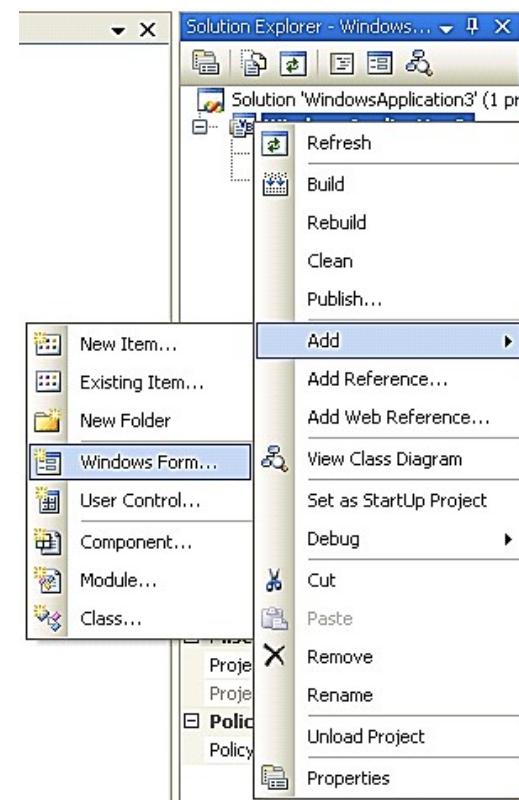


Creating DialogBox at design time

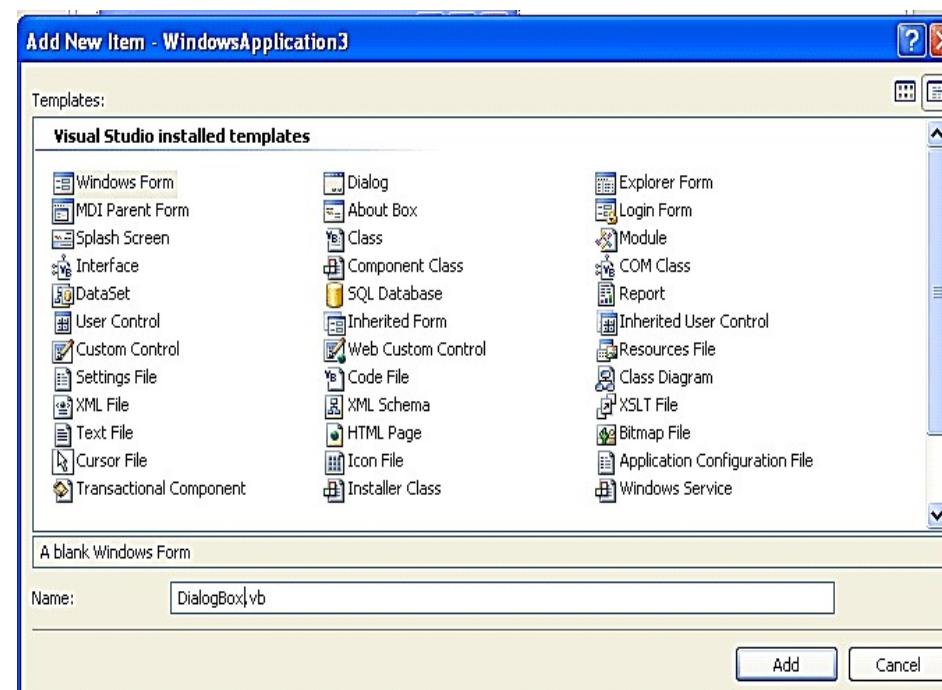
Following are the steps for creating a dialog box.

1. Add a form to your project by right-clicking the project in **Solution Explorer**, after that click **Add**, and then clicking **Windows Form**.

NOTES

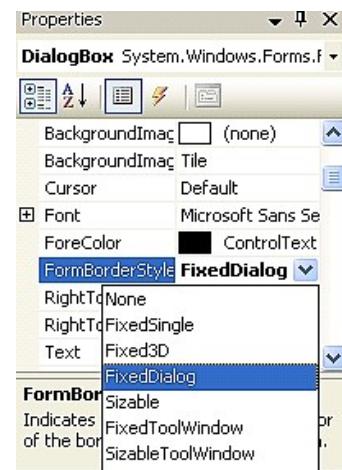


2. Right-click the form in **Solution Explorer** and choose **Rename**. Rename the form “DialogBox.vb”.



3. In the Properties window, change the FormBorderStyle property to **FixedDialog**.

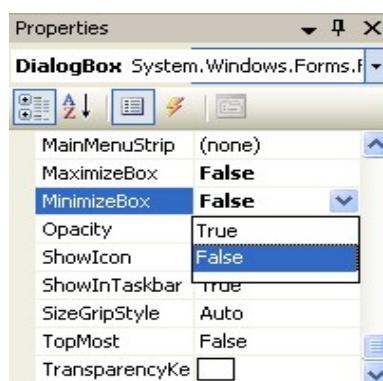
Windows Forms



NOTES

4. Customize the appearance of the form as needed.
5. Set the **ControlBox**, **MinimizeBox**, and **MaximizeBox** properties to false as shown below.

Dialog boxes do not usually include menu bars, window scroll bars, Minimize and Maximize buttons, status bars, or sizable borders.



NOTES**6. Customize event methods in the Code Editor.**

```

Public Class DialogBox
    Private Sub DialogBox_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles MyBase.Load
        End Sub
        End Sub
    End Class

```

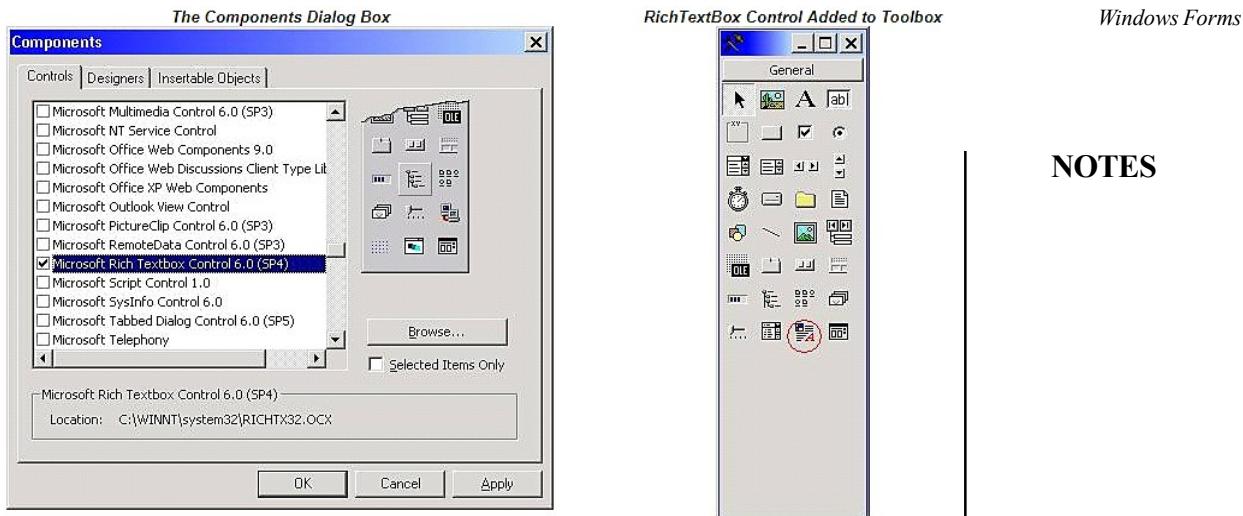
Table 6.1 Commonly Used DialogBox Controls

S. No.	Control & Description
1	ColorDialog It is a common dialog box that enables users to select a list of predefined colors along with controls that enable the users to specify custom colors.
2	FontDialog It is used to choose a typeface and style of the font from the pre-installed on the local computer and lets the user select the font, font size, and color.
3	OpenFileDialog It is used to open a file and allows the user to select a file to open.
4	SaveFileDialog It is used to select a location for saving a file and allows the user to specify the name of the file to save data.
5	PrintDialog This is used to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

6.7 RICHTEXTBOX

A RichTextBox control enables users to display or edit content flow along with advanced text formatting features than the conventional TextBox control that includes paragraphs, images, tables and much more. It allows users to apply formatting to any portion of text within the control. The selected text in the control can be assigned character and paragraph formatting. The properties provided by the control allows to make the selected text bold or italic, change in color, create subscripts and superscripts. Although, both the controls, RichTextBox and TextBox allow users to edit text, the two controls are used for different purposes. A TextBox is ideal when dealing with editing plain text. Whereas RichTextBox is best suitable when editing formatted text, tables, images, or other rich components.

The RichTextBox control must be added to the TextBox through Component dialog box as **Project -> Components** as shown in the screenshot below. The control is added to the ToolBox (as shown below in circle) when “Microsoft Rich TextBox Control 6.0” is checked and click ok.



RichTextBox also enables users to save and load RTF (Rich Text File) format and standard ASCII format. The following code demonstrate how RichTextBox is created.

```
public void CreateMyRichTextBox()
{
    RichTextBox richTextBox1 = new RichTextBox();
    richTextBox1.Dock = DockStyle.Fill;
    richTextBox1.LoadFile("C:\\\\MyDocument.rtf");
    richTextBox1.Find("Text",
    RichTextBoxFinds.MatchCase);

    richTextBox1.SelectionFont = new Font("Verdana",
    12, FontStyle.Bold);
    richTextBox1.SelectionColor = Color.Red;

    richTextBox1.SaveFile("C:\\\\MyDocument.rtf",
    RichTextBoxStreamType.RichText);

    this.Controls.Add(richTextBox1);
}
```

Table 6.2 Properties and their Purpose Associated with the RichTextBox

Properties	Purpose
AutoWordSelection	It is used to specify whether automatic word selection is ON or OFF. It has Boolean value. Default value is false.
BackColor	It is used to get or set background color of the RichTextBox.
ContextMenuStrip	It is used to specify the name of shortcut menu that is displayed when user right clicks on the RichTextBox.
DetectUrls	It is used to specify whether URL that is entered in RichTextBox is automatically displayed as hyperlink or not. It has Boolean value. Default value is true.

NOTES

NOTES

Enabled	It is used to specify whether RichTextBox control is enabled or not at run time. It has Boolean value. Default value is true.
Font	It is used to set Font Face, Font Style, Font Size and Effects of the text associated with RichTextBox control.
ForeColor	It is used to get or set Fore color of the text associated with RichTextBox control.
HideSelection	It is used to specify whether text selection should be hidden or not when RichTextBox lose its focus. It has Boolean value. Default value is true.
MaxLength	It is used to get or set maximum number of characters that can be entered in RichTextBox. Default value is 2147483647.
Multiline	It is used to specify whether RichTextBox can be expanded to enter more than one line of text or not. It has Boolean value. Default value is true.
ReadOnly	It is used to specify whether text associated with RichTextBox is ReadOnly or not. It has Boolean value. Default value is false.
RightMargin	It is used to get or set right margin of the text in RichTextBox.
ScrollBars	It is used to get or set type of scrollbars to be added with RichTextBox control. It has following 4 options:(1)None(2)Horizontal(2)Vertical(3)BothDefault value is Both.
Size	It is used to get or set height and width of RichTextBox control in pixel.
Text	It is used to get or set text associated with RichTextBox control.
TabIndex	It is used to get or set Tab order of the RichTextBox.
TabStop	It is used to specify whether user can use TAB key to set focus on RichTextBox control or not. It has Boolean value. Default value is true.
Visible	It is used to specify whether RichTextBox control is visible or not at run time. It has Boolean value. Default value is true.
WordWrap	It is used to specify whether line will be automatically word wrapped while entering multiple line of text in RichTextBox control or not. It has Boolean value. Default value is true.
ZoomFactor	It is used to get or set current zoom level of RichTextBox.
SelectedRtf	It is used to get or set currently selected RTF formatted text in RichTextBox.
SelectedText	It is used to get or set currently selected text in RichTextBox.
SelectionAlignment	It is used to get or set horizontal alignment of the text selected in RichTextBox.
SelectionBackColor	It is used to get or set BackColor of the text selected in RichTextBox.
SelectionBullet	It is used to get or set value which determines whether bullet style should be applied to selected text or not.
SelectionColor	It is used to get or set Fore Color of the text selected in RichTextBox.
SelectionFont	It is used to get or set font face, font style, and font size of the text selected in RichTextBox.

SelectionLength	It is used to get or set number of characters selected in the RichTextBox.	Windows Forms
SelectionStart	It is used to get or set starting point of the text selected in the RichTextBox.	

Table 6.3 Showing Various Method Associated with RichTextBox

Methods	Purpose
Cut	It is used to move current selection of RichTextBox into clipboard.
Copy	It is used to copies selected text of RichTextBox in clipboard.
Paste	It is used to replace current selection of TextBox by contents of clipboard. It is also used to move contents of Clipboard to RichTextBox control where cursor is currently located.
Select	It is used to select specific text from RichTextBox.
SelectAll	It is used to select all text of RichTextBox.
DeselectAll	It is used to deselect all text selected in RichTextBox.
Clear	It is used to clear all text from RichTextBox control.
AppendText	It is used to append text at the end of current text in RichTextBox control.
ClearUndo	It is used to clear information from the undo buffer of the RichTextBox.
Find	It is used to find starting position of first occurrence of a string in the RichTextBox control. If a string is not found then it returns -1.
SaveFile	It is used to save contents of RichTextBox in to Rich Text Format (RTF) file.
LoadFile	It is used to loads a Rich Text Format (RTF) file or standard ASCII text file into RichTextBox Control.
Undo	It is used to undo last edit operation of RichTextBox.
Redo	It is used to redo the last operation that is undo using undo method.

Table 6.4 Event Associated with RichTextBox

Event	Purpose
TextChanged	It is the default event of RichTextBox Control. It fires each time a text in the RichTextBox control got changed.

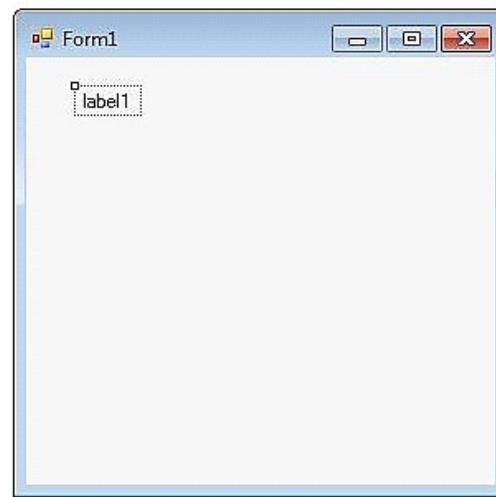
6.8 LABEL CLASS

The Label controls are the most frequently used VB control that acts as a medium to display some informative (instruction) text on the forms which does not change during runtime. These controls do not capture any mouse or keyboard events and do not participate in user inputs. The Label class is defined in the System.Windows.Forms namespace.

There are two ways of creating labels i.e. design time and run time.

NOTES

During design time, the user can drag the label control from the tool box and drop it on the form. The following screenshot shows the label on a form.

NOTES

The following code can also be written to create a label at the design time.

```
public void CreateMyLabel()
{
    // Create an instance of a Label.
    Label label1 = new Label();

    // Set the border to a three-dimensional border.
    label1.BorderStyle      =
System.Windows.Forms.BorderStyle.FixedSingle;
    // Set the ImageList to use for displaying an image.
    label1.ImageList = imageList1;
    // Use the second image in imageList1.
    label1.ImageIndex = 1;
    // Align the image to the top left corner.
    label1.ImageAlign = ContentAlignment.TopLeft;

    // Specify that the text can display mnemonic
    // characters.
    label1.UseMnemonic = true;
    // Set the text of the control and specify a mnemonic
    // character.
    label1.Text = "First &Name:";

    /* Set the size of the control based on the
    PreferredHeight and PreferredWidth values. */
    label1.Size = new Size (label1.PreferredWidth,
```

```
label1.PreferredHeight);
```

Windows Forms

```
//...Code to add the control to the form...
}
```

The following code shows how a label can be created at the run time. The creation of a label at the run time involves three steps. The first step is to create the instance of the Label Class, then set the properties of a label control and finally add the control to the form by calling Form.Controls.Add method. The following code snippet will add a label control at the run time

```
Dim dynamicLabel As New Label()
dynamicLabel.Name = "DynamicLabel"
dynamicLabel.BackColor = Color.Red
dynamicLabel.ForeColor = Color.Blue
dynamicLabel.Text = "I am a Dynamic Label"
dynamicLabel.Font = New Font("Georgia", 16)
Me.Controls.Add(dynamicLabel)
```

One can explore various properties associated with a label that appears in the Properties window. It provides all the functionality of Label control as it is derived from Label control. Similar to label control, it neither participate in user input or capture mouse or keyboard events.

NOTES

6.9 LINKLABEL CONTROL

In VB.Net, LinkLabel is a new standard control that allows to embed web style links in a form i.e. it provides the functionality of hyperlink in window application. This control provides all the functionality of label control as it is derived from it. Table 6.5 shows the properties associated with LinkLabel controls in VB.Net

Table 6.5 Properties Associated with LinkLabel Controls

Event	Purpose
LinkColor	It is used to get or set Fore color of the hyperlink in its default state.
ActiveLinkColor	It is used to get or set Fore color of the hyperlink, when user clicks it.
DisabledLinkColor	It is used to get or set Fore color of the hyperlink, when LinkLabel is disabled.
VisitedLinkColor	It is used to get or set Fore color of the hyperlink, when LinkVisited property of LinkLabel is set to true.
LinkVisited	It is used to specify whether hyperlink is already visited or not. It has Boolean value. Default value is false.
Text	It is used to get or set text associated with LinkLabel control.

NOTES	TextAlign	It is used to get or set alignment of the text associated with LinkLabel control.
	ForeColor	It is used to get or set Fore Color of the text associated with LinkLabel control.
	BackColor	It is used to get or set background color of the LinkLabel control.
	Enabled	It is used to specify whether LinkLabel control is enabled or not at runtime. It has Boolean value true or false. Default value is true.
	Visible	It is used to specify whether LinkLabel control is visible or not at runtime. It has Boolean value true or false. Default value is true.

Table 6.6 Showing the Various Methods Associated with LinkLabel Control

Event	Purpose
Show	It is used to Show LinkLabel Control at runtime.
Hide	It is used to Hide LinkLabel Control at runtime.
Focus	It is used to set input focus on LinkLabel Control.

Table 6.7 Event associated with LinkLabel.

Event	Purpose
Link Clicked	It is the default event of LinkLabel Control. It fires each time a user click on a hyperlink of LinkLabel Control.

LinkLabel control can also be created in two ways i.e. at design time and run time

The following example demonstrates the use of LinkLabel class. The example handles the LinkClicked event by opening a Web site.

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.LinkLabel linkLabel1;

    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        // Create the LinkLabel.
        this.linkLabel1 = new
```

```

System.Windows.Forms.LinkLabel();

        // Configure the LinkLabel's location.
        this.linkLabel1.Location = new
System.Drawing.Point(34, 56);
        // Specify that the size should be automatically
determined by the content.
        this.linkLabel1.AutoSize = true;

        // Add an event handler to do something when
the links are clicked.
        this.linkLabel1.LinkClicked+=new
System.Windows.Forms.LinkLabelLinkClickedEventHandler(this.
linkLabel1_LinkClicked);

        // Set the text for the LinkLabel.
this.linkLabel1.Text = "Visit Microsoft";

        // Set up how the form should be displayed and
add the controls to the form.
        this.ClientSize = new System.Drawing.Size(292,
266);
        this.Controls.AddRange(new
System.Windows.Forms.Control[] { this.linkLabel1 });
        this.Text = "Simple Link Label Example";
    }

    private void linkLabel1_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
    // Specify that the link was visited.
    this.linkLabel1.LinkVisited = true;

    // Navigate to a URL.
    System.Diagnostics.Process.Start("http://
www.microsoft.com");
}
}

```

Windows Forms

NOTES

6.10 PASSING FORMS

In VB.Net, it is possible for users to pass data and values between different forms. In C# and vb.net, there are numerous circumstances the new software engineers face a similar issue about how to pass information and qualities starting with one structure then onto the next. We can pass esteem starting with one structure then

NOTES

onto the next in various ways. Here you can see one of the most effortless technique to pass esteem starting with one structure then onto the next.

So as to impart between the forms, we are utilizing the Forms constructor to send these qualities. Constructors performs instate the information individuals from the new article and it has a similar name of the class. Here, we send the qualities as contentions of the constructor.

VB .Net

```
Public Sub New(ByVal sTitle As String, ByVal sText As
String)
    InitializeComponent()
    Me.Text = sTitle
    Me.Label1.Text = sText
End Sub
```

In the above code, you can see `InitializeComponent()` method. This method is automatically created and managed by Windows Forms designer and it defines everything you see on the form. It is better not to modify the `InitializeComponent` method.

Passing values step by step

1. Open a new project and drag two text boxes and a button in the Form1.



2. Add another form (Form2) in the project and add Label control on it.
3. Passing these two TextBox values from Form1 to Form2.
4. Here, call the constructor of Form2 and pass these values and set these values in form2.

Form1

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim sTitle As String
        Dim sText As String
        sTitle = TextBox1.Text
        sText = TextBox2.Text
        Dim frm As New Form2(sTitle, sText)
        frm.Show()
    End Sub
End Class
```

Form2

```
Public Class Form2
    Public Sub New(ByVal sTitle As String, ByVal sText As String)
        InitializeComponent()
        Me.Text = sTitle
        Me.Label1.Text = sText
    End Sub
End Class
```

NOTES**Check Your Progress**

5. What is the use of MessageBox in forms?
6. Write a note on RichTextBox control.
7. What are label controls?

6.11 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. WinForms is a Graphical User Interface (GUI) class library included in a .Net Framework. The primary purpose of providing this is to make the development of applications for desktop, tablet or PCs simpler and easier.
2. Main window or editor window, solution explorer window and properties window are the three main parts of WinForms window.
3. Multiple Document Interface or MDI Application provides a programming interface to create applications that enables to display multiple document at the same time, while displaying each documents in its own window.

NOTES

4. Mouse and keyboard events are the two types of events.
5. A message box is a type of dialog box that is used to communicate any piece of information with the user. `MessageBox` control in Windows Forms is used to display a message with the given text and action buttons.
6. A `RichTextBox` control enables users to display or edit content flow along with advanced text formatting features than the conventional `TextBox` control that includes paragraphs, images, tables and much more. It allows users to apply formatting to any portion of text within the control.
7. The Label controls are the most frequently used VB control that acts as a medium to display some informative (instruction) text on the forms which does not change during runtime. These controls do not capture any mouse or keyboard events and do not participate in user inputs.

6.12 SUMMARY

- Windows Forms is a Graphical User Interface (GUI) class library included in a .Net Framework. The primary purpose of providing this is to make the development of applications for desktop, tablet or PCs simpler and easier.
- Multiple Document Interface or MDI Application provides a programming interface to create applications that enable to display multiple document at the same time, while displaying each document in its own window.
- A windows application is an *event-driven* application where an event is a message sent by an object to inform an application about the occurrence of an action also known as event. It is a way through which user can interact with different element/ controls present in the form within the user interface, in any order they choose.
- A message box is a type of dialog box that is used to communicate any piece of information with the user. `MessageBox` control in Windows Forms is used to display a message with the given text and action buttons.
- An `InputBox` function displays a dialog box that prompts user to insert a value. The dialog box contains a text box for entering the value, OK and Cancel buttons and (optionally) a help button.
- `DialogBox` is a way to interact with users and retrieve information.
- A `RichTextBox` control enables users to display or edit content flow along with advanced text formatting features than the conventional `TextBox` control that includes paragraphs, images, tables and much more. It allows users to apply formatting to any portion of text within the control.
- The Label controls are the most frequently used VB control that acts as a medium to display some informative (instruction) text on the forms which does not change during runtime. These controls do not capture any mouse or keyboard events and do not participate in user inputs

- In VB.Net, `LinkLabel` is a new standard control that allows to embed web style links in a form i.e. it provides the functionality of hyperlink in window application.

Windows Forms

6.13 KEY WORDS

- **WinForms:** It is a graphical (GUI) class library included as a part of Microsoft.NET Framework or Mono Framework, giving a common platform to compose rich customer applications for workstation, work area, and tablet PCs.
- **Multiple Document Interface:** It is a Microsoft Windows programming interface for creating an application that enables users to work with multiple documents at the same time.
- **Event:** It is an action or occurrence detected by a program. It can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory.

NOTES

6.14 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Write the steps for creating a Windows Forms application.
2. Define an event.
3. Discuss the various types of events.
4. What is the use of `InputBox` in form? Write the general form of function.
5. Discuss the significance of label controls.

Long Answer Questions

1. What is MDI application? Write the steps for creating a parent and child form.
2. Explain the various types of mouse and keyboard events.
3. What is `MessageBox`? Explain, how you will create various types of `MessageBox`.
4. Write the steps for creating the `DialogBox` at design time.
5. Explain some properties, methods and event associated with `RichTextBox` control.
6. Explain the process of passing data and values between various forms.

6.15 FURTHER READINGS

NOTES

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

BLOCK - III

WINDOWS CONTROLS

UNIT 7 INTRODUCTION TO WINDOW CONTROLS

NOTES

Structure

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Commonly Used Controls
 - 7.2.1 Label (A) Control
 - 7.2.2 TextBox (abl) Control
 - 7.2.3 Button (ab) Control
 - 7.2.4 RadioButton (○) Control
 - 7.2.5 CheckBox (✓) Control
 - 7.2.6 ListBox (■) Control
 - 7.2.7 ComboBox (■) Control
 - 7.2.8 CheckedListBox (■) Control
 - 7.2.9 HScrollBar (◀▶) and VScrollBar (▲▼) Controls
 - 7.2.10 DateTimePicker (📅) Control
 - 7.2.11 Timer (⌚) Control
 - 7.2.12 Panel
 - 7.2.13 Splitters
 - 7.2.14 Track Bars
 - 7.2.15 Notify Icons
- 7.3 Menu Controls
- 7.4 Answers to Check Your Progress Questions
- 7.5 Summary
- 7.6 Key Words
- 7.7 Self Assessment Questions and Exercises
- 7.8 Further Readings

7.0 INTRODUCTION

In the previous unit, you have learnt about the Window forms. In this unit, you will learn about the commonly used Window controls and menu controls. Visual Basic (VB).NET offers Windows Forms which are the Graphical User Interfaces (GUIs) that allow building Windows-based applications. They provide you easier access to a number of different controls, including text boxes, radio buttons, check boxes, lists, drop-down menus, and buttons to create Windows applications quickly.

7.1 OBJECTIVES

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

NOTES

- Discuss the commonly used controls in VB.NET
- Explain how to add menus to a form
- Understand the properties of toolbar control

7.2 COMMONLY USED CONTROLS

VB .NET offers a wide variety of built-in controls that can be placed on the form to develop different applications. For example, we have controls for displaying information to user, allowing user input of text, and working with numbers. In addition, there are controls for the users to select among various choices. All the controls are placed in the Toolbox (see Figure 7.1) from where you can add it to a form. Each control can be manipulated by the properties and the events associated with it, which are displayed in the Properties window.

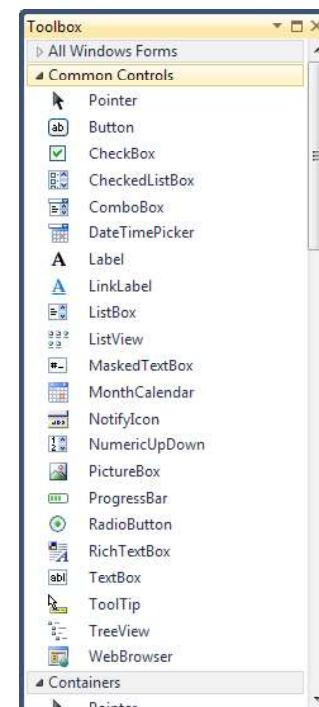


Fig. 7.1 Toolbox in VB 2010

The most commonly used control in VB is Pointer (☞) which is used to move or resize the controls placed on the form. It is automatically activated as soon as a control is placed on the form. Some other commonly used controls in VB are discussed in the following sections:

7.2.1 Label (A) Control

This control is used to display text on the form that should not be editable by the user at run-time unless we change its property. It is often used to provide descriptive caption text for other controls to help identify their purpose or to display results of some computation to the user. Some of the useful properties of the Label control are listed here.

- **Name:** To assign a name to the label. This name is used to refer to the label.
- **Text:** To specify the text to be displayed in the label.
- **Font:** To set the font, font size, and font style of the text to be displayed.
- **ForeColor:** To set the foreground color for the text to be displayed.
- **BackColor:** To set the background color of the label.
- **Margin:** To specify the margins of label from the top, left, right and bottom side of form.
- **Image:** To load an image in the label.
- **Size:** To set the height and width of label in pixels.
- **TextAlign:** To specify the position of label text. You have nine alignment options to choose from.
- **BorderStyle:** To specify the border style of the label.
- **Visible:** To specify whether the label should appear on the form at the run-time.

Most properties of the Label control can be set at design time in the Properties window or can be specified at run-time through programming in the Code Editor window. For example, if we have a label named Label1 and we want to specify its Text property as “My First Label Control”, we can do so at run-time using the following statement:

```
Label1.Text="My First Label Control"
```

Some of the events associated with the Label control are listed here:

- **Click:** This event occurs when you click a Label control.
- **DoubleClick:** This event occurs when you double-click a Label control.

7.2.2 TextBox (abl) Control

This control is used to take input as well as display output to the user. You can enter text as well as numeric values in a TextBox. By default, the TextBox control accepts only a single line of text; however, it can be made to accept multi-line text by setting appropriate property. In addition, you can also disable text editing by the user at the run-time (enabled, by default), specify the text limit, or add ScrollBar to the control. Some of the commonly used properties of the TextBox control are listed here.

NOTES

NOTES

- **Name:** To assign a name to the TextBox.
- **Font:** To set the font, font size, and font style of the text to be displayed.
- **MaxLength:** To specify the maximum length of the text that can be entered in the TextBox. A zero value in this field makes the length unlimited.
- **Multiline:** To make the TextBox accept the multiple lines of text.
- **ScrollBar:** To specify which ScrollBar (horizontal, vertical or both) should be displayed in a multiline TextBox.
- **Enabled:** To determine whether the text can be entered in the TextBox or not. A *True* value allows you to enter, while a *False* value restricts you from entering the text.
- **PasswordChar:** To hide the text with special characters (“*” is usually used). This helps the user to securely enter the information in the TextBox.
- **TextAlign:** To specify the alignment of text in the TextBox.
- **ReadOnly:** To specify whether the contents of a TextBox can be changed at the run-time.

Some of the events associated with the TextBox control are listed here.

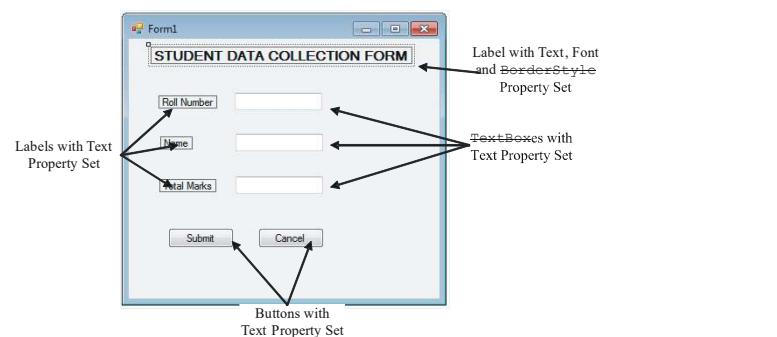
- **TextChanged:** This event occurs whenever you change the Text property of TextBox control.
- **Click:** This event occurs when you click the TextBox control.
- **DoubleClick:** This event occurs when you double-click the TextBox control.

7.2.3 Button (ab) Control

This control is same as that of CommandButton control used in earlier versions of VB. It is an interactive component that enables an application to perform some action on the form. When the button is clicked, it appears as if it is actually being pressed. Some of the commonly used properties of the Button control are listed here.

- **Name:** To assign a name to the button.
- **Text:** To specify the text that appears on the button.
- **Enabled:** To determine whether the button responds to an event or not. A *True* or *False* value is used to enable or disable this property, respectively.
- **Font:** To set the font, font size and font style of the text to be displayed on the button.

Like Label and TextBox controls, you can set the foreground and background color of Button control, add an image to it, specify the alignment of text displayed in the button, etc. Figure 7.2 shows a simple form designed using Label, TextBox and Button controls.

**NOTES****Fig. 7.2** A Form designed using Label, TextBox and Button Controls

The most common event associated with the Button control is `Click`, which occurs when the Button control is clicked. Notice that the Button control does not support double click event. In case any user attempts to double-click a button during execution of application, the `Click` event handler is actually invoked twice if the button is visible and available after the first click.

7.2.4 RadioButton (○) Control

This control is also referred to as **option button**. It is used to enable a user to select an option from a set of two or more mutually exclusive options. By mutually exclusive options we mean that a user can select only one option from the group of RadioButton. When the user selects a RadioButton in a group, other radio buttons get deactivated automatically. A RadioButton comprises two parts: a CheckBox portion (circle) and a caption. The CheckBox part indicates whether the RadioButton is selected (denoted by presence of a dot inside the circle) or cleared (denoted by empty circle). The caption part is used to assign a title on the RadioButton.

Some of the commonly used properties of RadioButton control are listed here.

- **Text:** To specify the text that appears inside the RadioButton.
- **Appearance:** To specify whether the RadioButton should appear as normal or as a Windows push button.
- **CheckAlign:** To specify the location of CheckBox portion (circle) of a RadioButton.
- **Checked:** To indicate whether the RadioButton is selected or cleared.
- **AutoCheck:** To specify whether the status (value of Checked property) and appearance (value of Appearance property) of a RadioButton should automatically change when a user clicks anywhere on it.
- **Tabstop:** To specify whether the user can use the **Tab** key to give focus to the RadioButton.

NOTES

Common events associated with RadioButton control are as follows.

- **CheckedChanged:** This event occurs when the value of Checked property of a RadioButton control changes.
- **AppearanceChanged:** This event occurs when the value of Appearance property of a RadioButton control changes.

7.2.5 CheckBox () Control

This control is similar to RadioButton control in a way as it also provides the users a group of options to choose from. However unlike RadioButton control, options are not mutually exclusive, that is, a user can select more than one option from the group and on selecting one option, and other options do not get deactivated. A CheckBox control also comprises two parts: a CheckBox part (indicated by a square box) and a caption part. Whenever, a user selects a specific CheckBox, a tick mark (✓) appears inside the square box, while an empty square box indicates that the option is not selected. A selected CheckBox holds the *True* value, while a cleared CheckBox holds the *False* value.

Besides the checked and unchecked states, a CheckBox can be in an indeterminate state in which the CheckBox is grayed out. This state is used to represent a partial or unknown selection. You can enable the indeterminate state of a CheckBox by setting its ThreeState property to *True*.

Some common properties of CheckBox control are as follows:

- **Text:** To specify the caption for the CheckBox.
- **Appearance:** To specify whether the CheckBox should appear as normal or as a Windows push button.
- **CheckAlign:** To specify the horizontal and vertical alignment of the check mark on a CheckBox.
- **Checked:** To indicate whether the CheckBox is selected or cleared.
- **CheckState:** To specify the state of a CheckBox.
- **AutoCheck:** To specify whether the value of Checked or CheckState property and appearance (value of Appearance property) of a CheckBox should automatically change when it is selected by a user.
- **ThreeState:** To specify whether to allow three states of a CheckBox instead to two.

Common events associated with CheckBox control are as follows:

- **CheckedChanged:** This event occurs when the value of Checked property of a CheckBox control changes.
- **CheckStateChanged:** This event occurs when the value of CheckState property of a CheckBox control changes.

- **AppearanceChanged:** This event occurs when the value of Appearance property of a CheckBox control changes.

Figure 7.3 shows a form illustrating the use of RadioButton and CheckBox controls.

NOTES

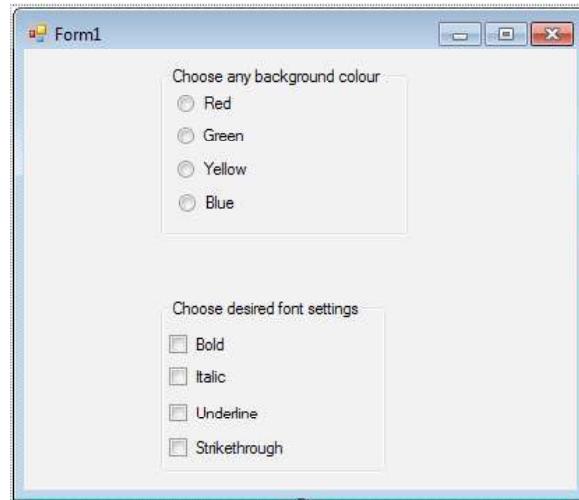


Fig. 7.3 A Form Designed with Radio Buttons and Check Boxes

Note: You can group multiple Radio Buttons or multiple Check Boxes using the **GroupBox** container control.

7.2.6 **ListBox** (>List Box) Control

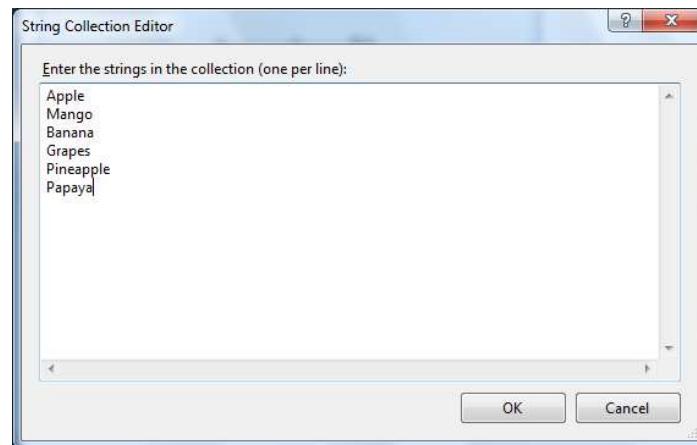
This control is used to display a list of choices which the user can select from. List boxes are used in situations where we have to display a large number of choices. In case the number of items to be displayed in the **ListBox** exceeds the height of **ListBox** control, a **ScrollBar** appears automatically that enables the user to scroll through the list. If you want a **ScrollBar** to always appear in the **ListBox** regardless of how many items are in it, you can do so by setting the **ScrollAlwaysVisible** property of **ListBox** to *True*. You can also make the items to appear in the **ListBox** horizontally in columns by setting its **MultiColumn** property to *True*. Some other common properties of **ListBox** control are listed here.

- **BorderStyle:** To specify what type of border is drawn around the **ListBox**.
- **ColumnWidth:** To specify the width of each column in a multicolumn **ListBox**.
- **ItemHeight:** To specify the height (in pixels) of items in a fixed-height user-drawn **ListBox**.
- **Items:** To return a list of items of a **ListBox**.

NOTES

- **Text:** To search the text of the selected item in a `ListBox`.
- **HorizontalScrollBar:** To specify whether or not to display a horizontal `ScrollBar` in the `ListBox` to view the items beyond the right edge of `ListBox`.
- **HorizontalExtent:** To specify the width (in pixels) by which a `ListBox` can be scrolled horizontally, that is, the horizontal scrolling area. This property is valid only if `HorizontalScrollBar` property is `True`.
- **SelectionMode:** To specify the methods used to select items in a `ListBox`. You can set the `SelectionMode` property of `ListBox` to `None` (no item can be selected), `One` (only a single item can be selected), `MultiSimple` (multiple items can be selected in a `ListBox`) or `MultiExtended` (multiple items can be selected from a `ListBox` using the `Ctrl`, `Shift` and/or `Arrow` keys).
- **SelectedItem:** To access the currently selected item in a `ListBox`.
- **SelectedItems:** To access the group of selected items in a `ListBox` in case the list allows multiple selections.
- **Sorted:** To specify whether the items in the `ListBox` should be sorted.
- **SelectedIndex:** To access the index of the selected item in the `ListBox`. The index value of items in the `ListBox` always starts with zero.
- **SelectedIndices:** To access the indices of selected items in a `ListBox` if the list allows multiple selections.
- **TopIndex:** To access the index of first visible item in the `ListBox`. Common events associated with `ListBox` control are as follows.
- **Click:** This event occurs when the `ListBox` control is selected.
- **SelectedIndexChanged:** This event occurs when the value of `SelectedIndex` property of a `ListBox` control changes.

You can add items in a `ListBox` at design time or run-time. At design time, you can add items in a `ListBox` using the `Items` property of `ListBox` control. For this, select the `ListBox` control on the form and click the `Ellipsis ()` button beside the `Items` property in the Properties window. This displays the String Collection Editor window (see Figure 7.4) where you can type the list items with one item per line. Press `Enter` key after typing each item before begin to type next item.



NOTES

Fig. 7.4 String Collection Editor Window

You can also add, remove or count items in a `ListBox` at run-time by using the methods of `ListBox` control. For example, you can count the number of items in the `ListBox` using the `Items.Count` method. You can add items in a `ListBox` using `Items.Add` or `Items.Insert` method. The difference between the two methods is that the former method is used to add item at the end of an unsorted `ListBox`, while the latter method is used to add item at a specific position in the `ListBox`. For example, if the name of a `ListBox` control is `Fruit_list` and we want to add item `Apple` to it, we can do so by writing the following statement in Code Editor Window.

```
Fruit_list.Items.Add("Apple")
```

In addition, you can use the `Items.Remove` method to delete the currently selected item, `Items.Clear` method to delete all the items at once and `Items.RemoveAt` method to delete item at a specific position in the `ListBox`.

7.2.7 ComboBox (❑) Control

This control, like `ListBox` control, is also used to present a large number of items. The difference is that the items of a `ComboBox` are displayed in a drop-down list. A `ComboBox` consists of two parts. The upper part of a `ComboBox` is a `TextBox` in which the user can type all or a part of a list item, while the lower part is a `ListBox` that displays the list of items for the user to select from. The user can select one or more items from the `ListBox` or can also enter data in a list through the keyboard.

Some common properties of a `ComboBox` control are listed here:

- **DropDownStyle:** To specify the style of drop-down part of `ComboBox`. You can choose from *Simple*, *DropDown*, or *DropDownList* styles. The *Simple* style specifies that the drop-

NOTES

down list is always displayed. The **DropDown** style is the default style which specifies that the **TextBox** part of **ComboBox** cannot be edited. In addition, one can view the drop-down list items by clicking the drop-down arrow button. The **DropDownList** style specifies that the **TextBox** part of **ComboBox** is editable and one must click drop-down arrow button to view the drop-down list.

- **DropDownWidth:** To specify the width of drop-down part of **ComboBox** in pixels.
- **DropDownHeight:** To specify the height (in pixels) of items in a fixed-height user-drawn **ListBox**.
- **MaxDropDownItems:** To specify the maximum number of items that can be displayed in the drop-down part of a **ComboBox**.
- **MaxLength:** To specify the maximum number of characters that can be entered by the user in the editable area (upper part) of a **ComboBox**.
- **Text:** To access the text associated with a **ComboBox**.
- **ItemHeight:** To specify the height (in pixels) of items in a **ComboBox**.
- **Items:** To retrieve a collection of items in a **ComboBox**.
- **SelectedItem:** To access the currently selected item in a **ComboBox**.
- **SelectedText:** To access the selected text in the **TextBox** part of a **ComboBox**.
- **Sorted:** To specify whether the items in the **ListBox** should be sorted.
- **SelectedIndex:** To access the index of the currently selected item in the **ComboBox**.

Common events associated with a **ComboBox** control are as follows.

- **DropDownStyleChanged:** This event occurs when the value of **DropDownStyle** property of a **ComboBox** control changes.
- **SelectedIndexChanged:** This event occurs when the value of **SelectedIndex** property of a **ComboBox** control changes.
- **DropDown:** This event occurs when the drop-down part of a **ComboBox** control appears.
- **SelectionChangeCommitted:** This event occurs when the selected item in the **ComboBox** changes and the change appears in the **ComboBox**.

7.2.8 CheckedListBox (checkbox) Control

This control is an extension of ListBox control that combines the capabilities of a ListBox with CheckBox control. It displays a list of items with a CheckBox on the left side of each list item. You can select one or more items from the ListBox by selecting respective check boxes. A checked ListBox is a better choice over ListBox when you want to display additional information about the list items being displayed.

Some common properties of a CheckedListBox control are listed here.

- **Multicolumn**: To specify whether or not the checked ListBox allows multiple columns.
- **Items**: To return a list of items of a checked ListBox.
- **Text**: To access the text of selected item in a checked ListBox.
- **CheckedItems**: To access the group of checked items in a checked ListBox.
- **CheckedIndices**: To access the checked indices in a checked ListBox.
- **Sorted**: To specify whether the items in the checked ListBox should be sorted.
- **SelectedIndex**: To access the index of selected item in a checked ListBox.
- **SelectedIndices**: To access the indices of selected items in a checked ListBox.
- **SelectedItem**: To access the currently selected item in a checked ListBox.
- **SelectedItems**: To access the group of selected items in a checked ListBox.
- **SelectionMode**: To specify the selection mode for items in a checked ListBox.

Common events associated with CheckedListBox control are as follows.

- **Click**: This event occurs when you click a CheckedListBox control.
- **SelectedIndexChanged**: This event occurs when the value of SelectedIndex property of a CheckedListBox control changes.
- **ItemCheck**: This event occurs when the checked state of an item in a checked ListBox changes.

NOTES

NOTES

Figure 7.5 shows a form designed with `ListBox`, `ComboBox` and `CheckedListBox` controls.

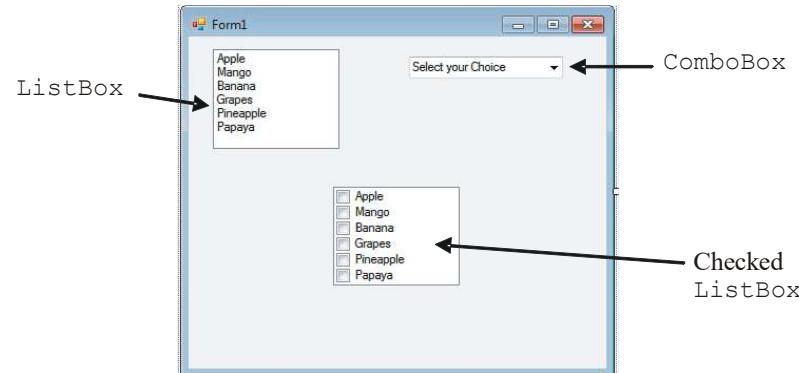


Fig. 7.5 A Form designed with `ListBox`, `ComboBox` and `Checked ListBox`

7.2.9 `HScrollBar` () and `VScrollBar` () Controls

The `HScrollBar` and `VScrollBar` controls are the two types of `ScrollBar` controls which are used to add horizontal and vertical `ScrollBars`, respectively. Each `ScrollBar` control displays a scroll box which can be scrolled horizontally or vertically depending on whichever control you have used. They are useful when we have a long list of items to navigate through.

Some of the properties of `HScrollBar` and `VScrollBar` controls are as follows:

- **AutoSize:** To specify whether the control should automatically resize to fit its contents.
- **Value:** To access the value corresponding to the current position of scroll box.
- **Maximum:** To specify the upper limit of scrollable range.
- **Minimum:** To specify the lower limit of scrollable range.
- **LargeChange:** To specify the value that is to be added to or subtracted from the `Value` property when the user clicks on the `ScrollBar` but outside the scroll box.
- **SmallChange:** To specify the value that is to be added to or subtracted from the `Value` property when the user clicks on the arrow button in the `ScrollBar`.

Some events associated with `HScrollBar` and `VScrollBar` controls are as follows:

- **Click:** This event occurs when the user clicks the `ScrollBar` control.

- **DoubleClick:** This event occurs when the user double clicks the ScrollBar control.
- **Scroll:** This event occurs when the user moves the scroll box through mouse or keyboard.
- **ValueChanged:** This event occurs when the Value property of ScrollBar control changes.

NOTES

Figure 7.6 shows a form with HScrollBar and VScrollBar controls.

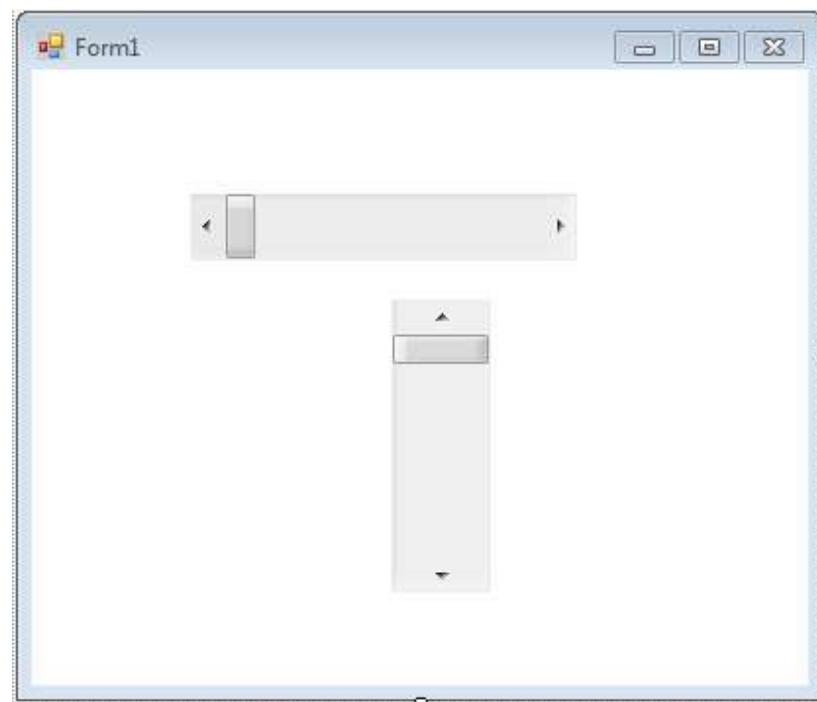


Fig. 7.6 A Form with Horizontal and Vertical ScrollBar Controls

7.2.10 DateTimePicker (📅) Control

This control allows you to display date and time. The user can also set a specific date and time by editing the current values displayed in the control. The DateTimePicker control consists of a TextBox with an accompanying drop-down calendar. When you click the drop-down arrow, a month calendar is displayed from which you can select the desired dates. Whatever date you select, the control validates the date and if sound valid, the date gets displayed in the TextBox part of control. The user can also simply type the desired date values in the TextBox part.

Some properties of DateTimePicker control are as follows.

- **Value:** To access the selected date and time. By default, this property is set to current date.
- **Text:** To specify the text that is to be displayed in the control.

NOTES

- **Format:** To specify the format of dates and times.
- **MaximumDateTime:** To specify the maximum date value of the DateTimePicker control.
- **MinimumDateTime:** To specify the minimum date value of the DateTimePicker control.
- **MaxDate:** To specify the maximum selectable date.
- **MinDate:** To specify the minimum selectable date.
- **CalenderFont:** To specify the font style for the calendar.
- **CalenderForeColor:** To specify the foreground color of the calendar.
- **CalenderTitleForeColor:** To specify the font color of the text in calendar title.
- **CalenderTitleBackColor:** To specify the background color of the calendar title.
- **CalenderMonthBackground:** To specify the background color of the calendar month.
- **Checked:** To determine whether the Value property of control holds a valid date-time value.
- **Padding:** To specify the distance between the contents of DateTimePicker control.
- **ShowCheckBox:** To specify whether a CheckBox should appear on the left side of the selected date.

Note that you can make the DateTimePicker control to display only time by setting its Format property to Time. In that case, the control does not show a drop-down calendar, but you can still use it to select only times that are valid.

Some events associated with DateTimePicker control are as follows.

- **DropDown:** This event occurs when the drop-down calendar appears.
- **FormatChanged:** This event occurs when the value of Format property changes.
- **ForeColorChanged:** This event occurs when the value of ForeColor property changes.
- **MouseClick:** This event occurs when user clicks on the control with mouse.
- **Paint:** This event occurs when the control is redrawn.
- **TextChanged:** This event occurs when the value of Text property changes.
- **ValueChanged:** This event occurs when the Value property of control changes.

Note: The current value of DateTimePicker control can be represented as a string using the ToString method of control.

7.2.11 Timer (⌚) Control

This control is used to generate periodic events. At design time, when you add a Timer control to your form, it does not appear on the form like other controls; rather it appears in the component tray below the form. Two common properties of Timer control are as follows.

- **Enabled:** To specify whether the timer is running or not.
- **Interval:** To specify the time interval between ticks of timer in milliseconds.

NOTES

The most common event on Timer control is Tick which occurs whenever the specified time interval passes by.

Figure 7.7 shows a form with DateTimePicker and Timer control.

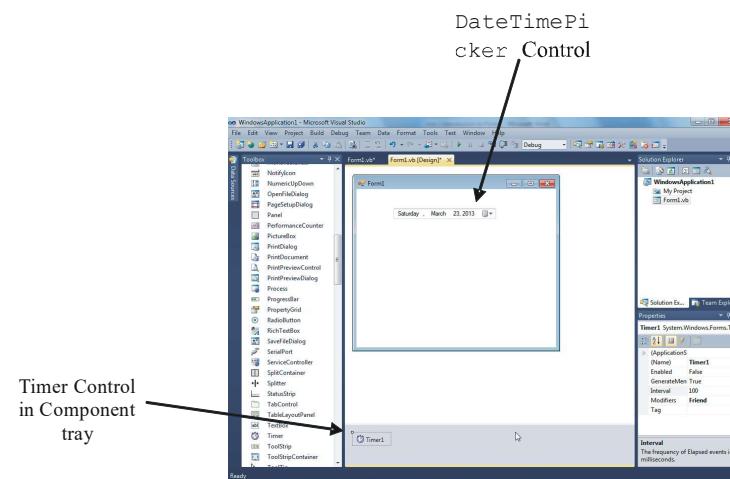


Fig. 7.7 A Form with DateTimePicker and Timer Control

Note: The timer can be started and stopped using the Start and Stop method of Timer control, respectively.

7.2.12 Panel

The Panel is a simplest container class. It presents the space in which an application can attach any other component. It inherits the Container class.

Class declaration

Following is the declaration for java.awt.Panel class:

```
public class Panel
    extends Container
    implements Accessible
```

NOTES

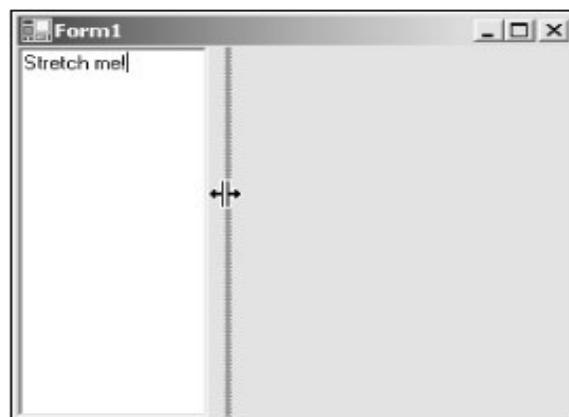
Example 7.1

```
import java.awt.*;
public class Example
{
    Example()
    {
        Frame f= new Frame("Example");
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        Button b2=new Button("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new Example();
    }
}
```

7.2.13 Splitters

Splitters are used to let the user resize controls.

Working: one can add a control to a form, and then dock it. Next, you add a splitter and dock it to the same side of the same container. It places the control just before the splitter in the docking order. When we run the program, the splitter is invisible until the mouse passes over it, when the splitter changes the mouse cursor, indicating that the control can be resized, as shown in Figure 7.8, which is the Splitters example on the CD-ROM. In this figure, we are moving the splitter, and when we release it, the control resizing (a text box in this case) is indeed resized to match.



NOTES

Fig. 7.8 A Splitter in Action

Properties of splitters

- Used to create complex user interfaces;
- A selection in one panel determines what objects are shown in the other panel.
- This arrangement is very effective for displaying and browsing information.
- Having two panels allow you to aggregate information in areas, and the bar, or "splitter," makes it easy for users to resize the panels.

Example 7.2: A control consisting of a movable bar that divides a container's display area into two resizable panels:

```
[System.Runtime.InteropServices.ComVisible(true)]  
[System.Windows.Forms.Docking(System.Windows.Forms.DockStyle.Fill)]  
[System.Runtime.InteropServices.ClassInterface(System.Runtime.InteropServices.ClassInterfaceType.AutoDispatch)]  
  
public class SplitContainer :  
    System.Windows.Forms.ContainerControl,  
    System.ComponentModel.ISupportInitialize
```

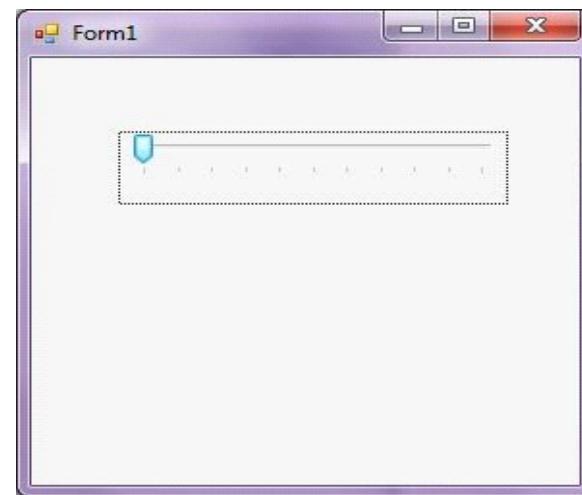
7.2.14 Track Bars

Track bar also known as Slider Control. It is used to navigate through a large amount of information or for visually adjusting a numeric setting. This control has two parts: the thumb, also known as a slider, and the tick marks.

Creating a TrackBar

- We can create a TrackBar control using a Forms designer at design-time or using the TrackBar class in code at run-time.

NOTES



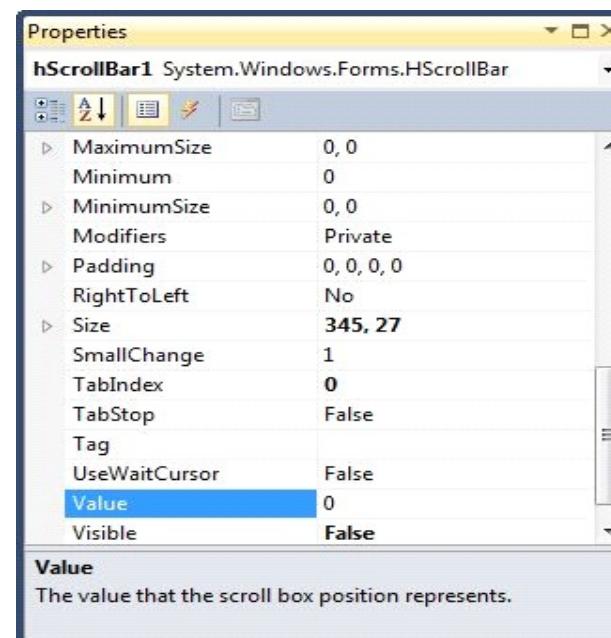
The following code snippet creates a TrackBar control object.

```
Dim tbar As New TrackBar
```

In the next step, you can set properties of a TrackBar control. The following code snippet sets the height and width properties of a TrackBar.

```
tbar.Height = 40  
tbar.Width = 300
```

Next step is to set TrackBar Properties.



7.2.15 NotifyIcons

It is used to add system tray notification functionality to a Windows Forms application. An icon will be added to the system tray when we will run the system and we can add double click or menus to the icon to take some actions.

Creating a NotifyIcon

It acts as a component in the background without visual representation. We can use NotifyIcon or Form designer to create a NotifyIcon. To add a NotifyIcon to a Windows Forms application, drag a NotifyIcon component to the Toolbox onto a Form.

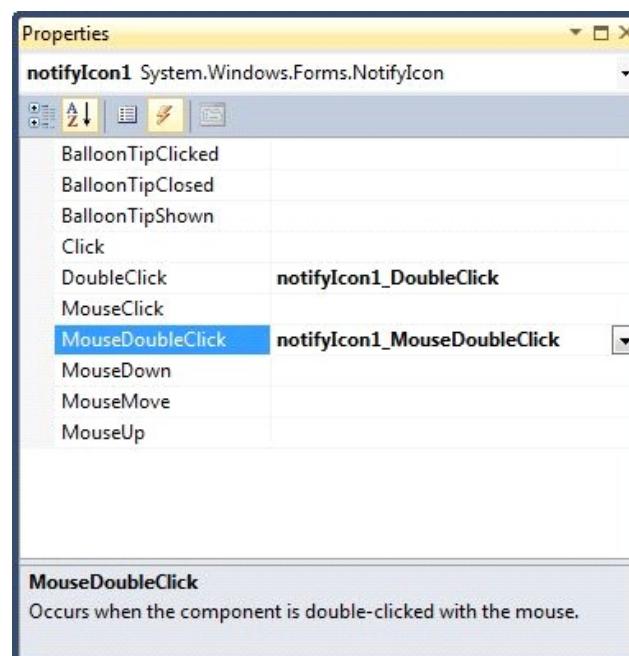
NOTES

After adding a NotifyIcon, first thing you want to do is to add an Icon that would be displayed in the icon tray. We can do this by clicking on little handle on the NotifyIcon and select Choose Icon link as shown in figure given below.



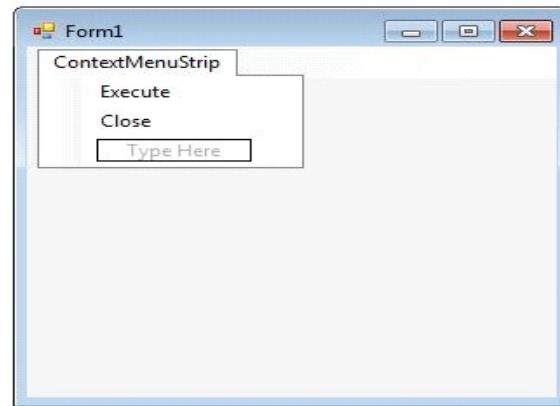
NotifyIcon Events

We will double click on an Icon to open an application user interface and right click on an icon to open a context menu. The screenshot below shows the events associated with a NotifyIcon and we are going to set DoubleClick event handler and on double click, we will open the application.



Before proceeding, we are going to add a ContextMenu using a ContextMenuStrip control that will be opened when right mouse click event is fired on the system tray icon. The ContextMenu is shown below.

NOTES



The code for the ContextMenu item event handlers is listed below were on Execute menu item, we are going to activate the application and on Close menu item, we are going to exit the application.

```
Private Sub ExecuteToolStripMenuItem_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ExecuteToolStripMenuItem.Click
    Me.Activate()
End Sub
```

```
Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click
    Me.Close()
End Sub
```

Setting ContextMenu

Now, we add the following code on the Form initialization to set ContextMenu of the NotifyIcon.

```
notifyIcon1.ContextMenuStrip = contextMenuStrip1;
```

Here is the mouse double click event handler where we simply activate the Window.

```
Private Sub NotifyIcon1_MouseDoubleClick(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) Handles NotifyIcon1.MouseDoubleClick
    If (Me.WindowState = FormWindowState.Minimized) Then
```

```

Me.WindowState = FormWindowState.Normal
Me.Activate()
End If
End Sub

```

*Introduction to
Window Controls*

NOTES

7.3 MENU CONTROLS

The menus are commands grouped by a common topic. Menus make it easy for the users to navigate the application, and to see the menus they have already used in Windows, as the File menu. As a bonus, using a menu to hold commands avoids using precious real estate on your forms.

Adding Menus to a Form

Exercise adding a menu to a Form at design time. In this, you will add a File menu with an Exit menu item to a form at design time. The steps are as follows:

- From the **File** menu, select **New Project**.
- In the **Visual Basic Projects** list, select the **Windows Application** template, and then click **OK**.
- Select **winforms** tab from the **Toolbox** and double-click on the **MainMenu** control. A menu will be added to the form showing the **textType** here.
- In the **Menu Designer**, click the text **Type Here** to select the menu, and then type and File.
- Click the area below the **File** menu item to add another entry to the same menu, and type Exit.

The Windows Forms framework includes four menu enhancements that you can use to convey information to the user. Table 7.1 describes these menu enhancements.

Table 7.1 Menu Enhancements

Enhancement	Description
Check marks	Check marks indicate whether a feature is turned on or off or to point out which list of files is being showed.
Shortcut keys	Allow access to menu items using keyboard commands.
Access keys	Allow keyboard navigation of menus (pressing the ALT key and the underlined access key choose the desired menu or menu item).
Separator bars	Used to group related commands within a menu and make menus easier to read.

Context Menus

Context menus are menus that appear when an item is right-clicked. In any windows application when you right-click your mouse, you get a menu which might display some shortcuts from the **Edit Menu**, for example, **Cut**, **Copy**, **Paste**, **Paste Special** and so on. All these menu items which are available when you right-click are called Context Menus. In Visual Basic, context menus are created with the **ContextMenu** component. The **ContextMenu** component is edited exactly the

NOTES

same way the **MainMenu** component is edited. The **ContextMenu** appears at the top of the form and you can add menu items by typing them. To associate a **ContextMenu** with a particular form or control you need to set the **ContextMenu** property of that form or control to the appropriate menu.

When you drag a **ContextMenu** control to a form, the IDE generates a code just like it did for main menu items. Consider the following example:

```
Friend WithEvents ContextMenu1 As
System.Windows.Forms.ContextMenu
Also, consider the following example:
Me.ContextMenu1 = New System.Windows.Forms.ContextMenu()
()
```

You can also use the useful **Edit Names** features for the context menus by right-clicking, in the order to quickly assign a value to the name property to the context menu item as well.

Adding ToolBars to Forms

A toolbar is an additional control that is frequently docked to an edge of a form. In the .NET Framework, toolbars show buttons that can appear as standard buttons, toggle-style buttons, or drop-down style buttons. Toolbars can also show drop-down menus that activate commands. Table 7.2 describes the properties of the Toolbar control:

Table 7.2 Properties of the Toolbar Control

Property	Description
(Bindings)	This collection holds all the bindings of properties of the toolbar to data sources.
(Name)	Indicates the name used in code to identify the toolbar.
AccessibleDescription	The description that will be reported to accessibility clients.
AccessibleName	The name that will be reported to accessibility clients.
AccessibleRole	The role that will be reported to accessibility clients.
AllowDrop	Determines if the toolbar will receive drag-and-drop notifications.
Anchor	The anchor of the toolbar.
Appearance	Controls the appearance of the toolbar.
AutoSize	Controls whether the toolbar will automatically size itself based on button size.
BackgroundImage	The background image used for the toolbar.
BorderStyle	Controls what type of border the toolbar will have.
Buttons	The collection of toolbar buttons that make up the toolbar.
ButtonSize	Suggests the size of buttons in the toolbar. Button sizes might still be different based on text, drop-down arrows, and others.
CausesValidation	Indicates whether the toolbar causes and raises validation events.
ContextMenu	The shortcut menu to display when the user right-clicks the toolbar.
Cursor	The cursor that appears when the mouse passes over the toolbar.
Divider	Controls whether the toolbar will display a 3D line at the top of its client area.
Dock	The docking location of the toolbar, indicating which borders are docked to the container.
DropDownArrows	Controls whether the toolbar will display an arrow on the side of drop-down buttons.
Enabled	Indicates whether the control is enabled.

Font	The font used to display text in the control.
ImageList	The ImageList control from which the toolbar will get button images.
IMEMode	Determines the IME status of the toolbar when selected.
Location	The position of the top-left corner of the toolbar with respect to its container.
Locked	Determines if the toolbar can be moved or resized.
Modifiers	Indicates the visibility level of the toolbar.
ShowToolTips	Indicates whether tool tips will be shown for each button, if available.
Size	The size of the toolbar in pixels.
TabIndex	Determines the index in the Tab order that the toolbar will occupy.
TabStop	Indicates whether the user can use the Tab key to give focus to the toolbar.
 TextAlign	Controls how the text is positioned relative to the image in each button.
Visible	Determines whether the toolbar is visible or hidden.
Wrappable	Indicates if more than one row of buttons is allowed.

NOTES

Check Your Progress

1. List the useful properties of the Label control.
2. What is the purpose of RadioButton control?
3. Define the term padding.

7.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Some useful properties associated with Label control are Name, Text, Font, ForeColor, BackColor, Margin, Image, Size, TextAlign, and BorderStyle.
2. RadioButton control is used to enable a user to select an option from a set of two or more mutually exclusive options. By mutually exclusive options we mean that a user can select only one option from the group of radio buttons.
3. Padding is the property of DateTimePicker control that is used to specify the distance between the contents of it.

7.5 SUMMARY

- VB .NET offers a wide variety of built-in controls that can be placed on the form to develop different applications. For example, we have controls for displaying information to user, allowing user input of text, and working with numbers.
- The most commonly used control in VB is Pointer control which is used to move or resize the controls placed on the form. It is automatically activated as soon as a control is placed on the form.

NOTES

- **Label (A)** control is used to display text on the form that should not be editable by the user at run-time unless we change its property. It is often used to provide descriptive caption text for other controls to help identify their purpose or to display results of some computation to the user.
- **TextBox ()** control is used to take input as well as display output to the user. You can enter text as well as numeric values in a TextBox.
- **Button ()** control is an interactive component that enables an application to perform some action on the form.
- **RadioButton ()** control is also referred to as option button. It is used to enable a user to select an option from a set of two or more mutually exclusive options.
- **CheckBox ()** control is similar to RadioButton control in a way as it also provides the users a group of options to choose from.
- **ListBox ()** control is used display a list of choices which the user can select from. List boxes are used in situations where we have to display a large number of choices.
- **ComboBox ()** control, like ListBox control, is also used to present a large number of items. The difference is that the items of a ComboBox are displayed in a drop-down list.
- **CheckedListBox ()** control is an extension of ListBox control that combines the capabilities of a ListBox with CheckBox control. It displays a list of items with a CheckBox on the left side of each list item.
- **HScrollBar ()** and **VScrollBar ()** controls are the two types of ScrollBar controls which are used to add horizontal and vertical ScrollBars, respectively.
- **DateTimePicker ()** control allows you to display date and time. The user can also set a specific date and time by editing the current values displayed in the control.
- **Timer ()** control is used to generate periodic events. At design time, when you add a Timer control to your form, it does not appear on the form like other controls; rather it appears in the component tray below the form.

7.6 KEY WORDS

- **Controls:** The objects which are placed or embedded on a form to receive input and display output.
- **Properties:** Describes the characteristics of a control.
- **Methods:** The actions that can be performed on the object.
- **Events:** The occurrence of user-generated actions applied on an object.

7.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Give a brief description of `TextBox` control.
2. What do you mean by `DropDownStyle` property?
3. Why do we use `DateTimePicker` control?

NOTES

Long Answer Questions

1. Write short notes on the following:
 - (a) `Button` control
 - (b) `RadioButton` control
 - (c) `ComboBox` control
 - (d) `CheckedListBox` control
 - (e) `Timer` control
2. Explain how will you create the splitters, pickers, track bars and notify icons.

7.8 FURTHER READINGS

Holzner, Steven. 2002. *Visual Basic .NET Black Book*, United Kingdoms: Coriolis Group Books.

Deitel, Harvey M. 2001. *Visual Basic.NET How to Program*, 2nd edition. New Jersey: Prentice Hall.

UNIT 8 TREE AND LISTVIEW

NOTES**Structure**

- 8.0 Introduction
 - 8.1 Objectives
 - 8.2 The ListView and TreeView Classes
 - 8.2.1 Toolbar
 - 8.2.2 Status Bar Control
 - 8.2.3 Progress Bar Control
 - 8.3 Answers to Check Your Progress Questions
 - 8.4 Summary
 - 8.5 Key Words
 - 8.6 Self Assessment Questions and Exercises
 - 8.7 Further Readings
-

8.0 INTRODUCTION

In this unit, you will learn about the ListView and TreeView control. ListView control is a window which is used to display list of items using different views that includes text, small and large images. TreeView is another window control used to display hierarchical list of items, such as directory hierarchy. You will also learn about the tab control, status bar control and progress bar control.

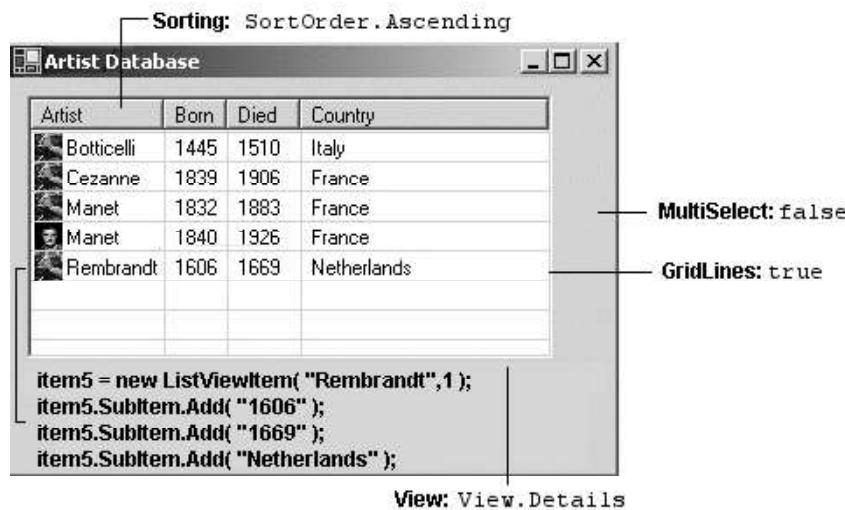
8.1 OBJECTIVES

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

- Understand the significance of ListView and TreeView control
 - Create and work with ListView and TreeView control
 - Work with Tab Control, status bar control and progress bar control
-

8.2 THE LISTVIEW AND TREEVIEW CLASSES

ListView is another window control that displays textual information as a collection of items in rows and columns with column heading. It allows user to create an interface like Window Explorer where each file can be shown by its name and its desired information about the size, last modified date etc. The items can also be displayed as a list with icons. There are several ways available to arrange and display items. It is the way of displaying data socially as items and sub items.

**NOTES****Fig. 8.1 An Example of ListView****Creating a ListView Object**

The ListView is created using a parameter-less constructor.

```
ListView listView1 = new ListView();
```

Defining Appearance of ListView Object

```
// Set the view to show details
listView1.View = View.Details;
```

The View property specifies one of the five layouts for the control.

1. Details. An icon and item's text are displayed in column one. Subitems are displayed in the remaining columns.
2. LargeIcon. A large icon is shown for each item with a label below the icon.
3. List. Each item is displayed as a small icon with a label to its right. The icons are arranged in columns across the control.
4. SmallIcon. Each item appears in a single column as a small icon with a label to its right.
5. Tile. Each item appears as a full-size icon with the label and subitem details to the right of it. Only available for Windows XP and 2003.

In ListView, View property can be changed at runtime to switch among the possible views. In fact, you can recognize that the view options correspond exactly to the view menu options available in Windows Explorer.

After the Details view is selected, other properties are set that define the control's appearance and behaviour.

```
// Allow the user to rearrange columns
listView1.AllowColumnReorder = true;
```

Tree and ListView

```
// Select the entire row when selection is made  
listView1.FullRowSelect = true;  
// Display grid lines  
listView1.GridLines = true;  
// Sort the items in the list in ascending order  
listView1.Sorting = SortOrder.Ascending;
```

NOTES

These properties automatically sort the items, permit the user to drag columns around to rearrange their order, and cause a whole row to be highlighted when the user selects an item.

Set Column Headers

In a detail view, data is not displayed until at least one column is added to the control. Its simplest form is:

```
ListView.Columns.Add(caption, width, textAlign)
```

Caption is the text to be displayed. Width specifies the column's width in pixels. It is set to -1 to size automatically to the largest item in the column, or -2 to size to the width of the header.

```
// Create column headers for the items and subitems  
listView1.Columns.Add("Artist", -2,  
HorizontalAlignment.Left);  
listView1.Columns.Add("Born", -2,  
HorizontalAlignment.Left);  
listView1.Columns.Add("Died", -2,  
HorizontalAlignment.Left);  
listView1.Columns.Add("Country", -2,  
HorizontalAlignment.Left);
```

The Add method creates and adds a ColumnHeader type to the ListView's Columns collection. The method also has an overload that adds a ColumnHeader object directly:

```
ColumnHeader cHeader:  
cHeader.Text = "Artist";  
cHeader.Width = -2;  
cHeader.TextAlign = HorizontalAlignment.Left;  
ListView.Columns.Add(ColumnHeader cHeader);
```

Create ListView Items

Several overloaded forms of the ListView constructor are available. They can be used to create a single item or a single item and its sub items. There are also options to specify the icon associated with the item and set the foreground and background colors.

Constructors:

Tree and ListView

```
public ListViewItem(string text);
public ListViewItem(string[] items );
public ListViewItem(string text,int imageIndex );
public ListViewItem(string[] items,int imageIndex );
public ListViewItem(string[] items,int imageIndex,
    Color foreColor,Color backColor,Font font);

The following code demonstrates how different overloads
can be used to create the items and sub items.

// Create item and three sub items
ListViewItem item1 = new ListViewItem("Manet",2);
item1.SubItems.Add("1832");
item1.SubItems.Add("1883");
item1.SubItems.Add("France");
// Create item and subitems using a constructor only
ListViewItem item2 = new ListViewItem
    (new string[] {"Monet","1840","1926","France"}, 3);
// Create item and subitems with blue background color
ListViewItem item3 = new ListViewItem
    (new string[] {"Cezanne","1839","1906","France"}, 1,
    Color.Empty, Color.LightBlue, null);

To display the items, add them to the Items collection
of the ListView control:
// Add the items to the ListView
listView1.Items.AddRange(
    new ListViewItem[]{item1,item2,item3,item4,item5});
```

NOTES

Specifying Icons

Two collections of images can be associated with a `ListView` control as `ImageList` properties. `LargeImageList` contains images used in the `LargeIcon` view; and `SmallImageList` contains images used in all other views. Think of these as zero-based arrays of images that are associated with a `ListViewItem` by the `imageIndex` parameter in the `ListViewItem` constructor. Even though they are referred to as icons, the images may be of any standard graphics format.

The following code creates two `ImageList` objects, adds images to them and assigns them to the `LargeImageList` and `SmallImageList` properties.

```
// Create two ImageList objects
ImageList imageListSmall = new ImageList();
ImageList imageListLarge = new ImageList();
imageListLarge.ImageSize = new Size(50,50); // Set
```

```

        image size
        // Initialize the ImageList objects
        // Can use same images in both collections since
        they're resized
        imageListSmall.Images.Add(Bitmap.FromFile("C:\\botti.gif"));
        imageListSmall.Images.Add(Bitmap.FromFile("C:\\cezanne.gif"));
        imageListLarge.Images.Add(Bitmap.FromFile("C:\\botti.gif"));
        imageListLarge.Images.Add(Bitmap.FromFile("C:\\cezanne.gif"));
        // Add other images here
        // Assign the ImageList objects to the ListView.
        listView1.LargeImageList = imageListLarge;
        listView1.SmallImageList = imageListSmall;
        ListViewItem lvItem1 = new ListViewItem("Cezanne", 1);

```

An index of 1 selects the *cezanne.gif* images as the large and small icons. Specifying an index not in the *ImageList* results in the icon at index 0 being displayed. If neither *ImageList* is defined, no icon will be displayed. Figure 8.2 shows the *ListView* from Figure 8.1 with its view set to *View.LargeIcon*.

```
listView1.View = View.LargeIcon;
```

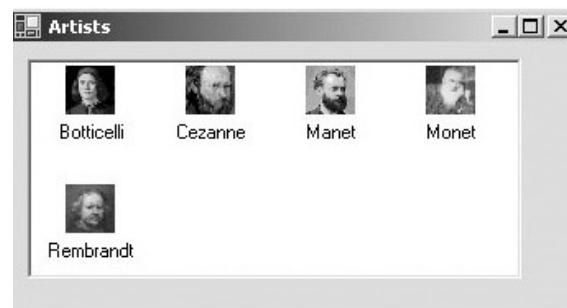


Fig. 8.2 ListView Set to Large Icon

Working with the *ListView* Control

Common tasks associated with the *ListView* control include iterating over the contents of the control, iterating over selected items only, detecting the item that has focus and sorting the items by any column. Following are code segments to perform these tasks.

Iterating over All Items or Selected Items

You can use *foreach* to create nested loops that select an item and then iterate through the collection of sub-items for the item in the outside loop.

```

foreach (ListViewItem lvi in listView1.Items)
{
    string row = "";
    foreach (ListViewItem.ListViewSubItem sub in

```

```

lvi.SubItems)
{
    row += " " + sub.Text;
}
MessageBox.Show(row); // List concatenated subitems
}

```

Tree and ListView

NOTES

There are number of things to be aware of when working with these collections. First, the first subitem (index 0) element actually contains the text for the item but not a subitem. Second, the ordering of subitems is not affected by rearranging columns in the ListView control. This changes the appearance but does not affect the underlying ordering of subitems.

The same logic is used to list only selected items (MultiSelect = true permits multiple items to be selected). The only difference is that the iteration occurs over the ListView.SelectedItems collection.

```
foreach (ListViewItem lvisel in listView1.SelectedItems)
```

Detecting the Currently Selected Item

In addition to the basic control events such as Click and DoubleClick, the ListView control adds a SelectedIndexChanged event to indicate when focus is shifted from one item to another. The following code implements an event handler that uses the FocusedItem property to identify the current item.

```

// Set this in the constructor
listView1.SelectedIndexChanged += 
    new EventHandler(lv_IndexChanged);
// Handle SelectedIndexChanged Event
private void lv_IndexChanged(object sender,
System.EventArgs e)
{
    string ItemText = listView1.FocusedItem.Text;
}

```

Note that this code can also be used with the Click events because they also use the EventHandler delegate. The MouseDown and MouseUp events can also be used to detect the current item. Given below is a sample code for MouseDown event handler.

```

private void listView1_MouseDown(object sender,
MouseEventArgs e)
{
    ListViewItem selection = listView1.GetItemAt(e.X,
e.Y);
    if (selection != null)
    {

```

```

        MessageBox.Show("Item Selected: "+selection.Text);
    }
}

```

NOTES

The `ListView.GetItemAt` method returns an item at the coordinates where the mouse button is pressed. If the mouse is not over an item, null is returned.

Sorting Items on a ListView Control

Sorting items in a `ListView` control by column values is a simple feature to implement. The secret to its simplicity is the `ListViewItemSorter` property that specifies the object to sort the items anytime the `ListView.Sort` method is called. Implementation requires three steps:

1. Set up a delegate to connect a `ColumnClick` event with an event handler.
2. Create an event handler method that sets the `ListViewItemSorter` property to an instance of the class that performs the sorting comparison.
3. Create a class to compare column values. It must inherit the `IComparer` interface and implement the `IComparer.Compare` method.

The following code implements the logic. When a column is clicked, the event handler creates an instance of the `ListViewItemComparer` class by passing it the column that was clicked. This object is assigned to the `ListViewItemSorter` property, which causes sorting to occur. (“Working with Objects in C#”).

```

// Connect the ColumnClick event to its event handler
listView1.ColumnClick += new
    ColumnClickEventHandler(ColumnClick);
// ColumnClick event handler
private void ColumnClick(object o, ColumnEventArgs
e)
{
    // Setting this property immediately sorts the
    // ListView using the ListViewItemComparer object
    this.listView1.ListViewItemSorter =
        new ListViewItemComparer(e.Column);
}
// Class to implement the sorting of items by columns
class ListViewItemComparer : IComparer
{
    private int col;
    public ListViewItemComparer()
    {
        col = 0; // Use as default column
    }
    public ListViewItemComparer(int column)

```

```

{
    col = column;
}
// Implement IComparer.Compare method
public int Compare(object x, object y)
{
    string xText = ((ListViewItem)x).SubItems[col].Text;
    string yText = ((ListViewItem)y).SubItems[col].Text;
    return String.Compare(xText, yText);
}
}

```

The TreeView Class

As the name implies, the `TreeView` control provides a tree-like view of hierarchical data as its user interface. Underneath, its programming model is based on the familiar tree structure consisting of parent nodes and child nodes. Each node is implemented as a `TreeNode` object that can in turn have its own `Nodes` collection. Figure 8.3 shows a `TreeView` control that is used in conjunction with a `ListView` to display enum members of a selected assembly.

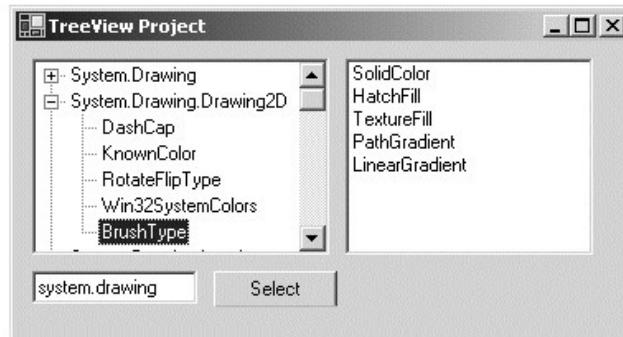


Fig. 8.3 Using TreeView Control (Left) and ListView (Right) to List Enum Values

The TreeNode Class

Each item in a tree is represented by an instance of the `TreeNode` class. Data is associated with each node using the `TreeNode`'s `Text`, `Tag`, or `ImageIndex` properties. The `Text` property holds the node's label that is displayed in the `TreeView` control. `Tag` is an object type, which means that any type of data can be associated with the node by assigning a custom class object to it. `ImageIndex` is an index to an `ImageList` associated with containing `TreeView` control. It specifies the image to be displayed next to the node.

In addition to these basic properties, the `TreeNode` class provides numerous other members that are used to add and remove nodes, modify a node's appearance, and navigate the collection of nodes in a node tree.

NOTES

Table 8.1 Selected Members of the TreeNode Class**NOTES**

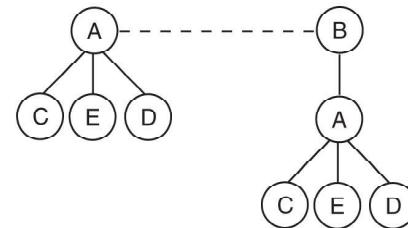
Use	Member	Description
Appearance	BackColor, ForeColor	Sets the background color and text color of the node.
	Expand(), Collapse()	Expands the node to display child nodes or collapses the tree so no child nodes are shown.
Navigation	FirstNode, LastNode, NextNode, PrevNode	Returns the first or last node in the collection. Returns the next or previous node (sibling) relative to the current node.
	Index	The index of the current node in the collection.
	Parent	Returns the current node's parent.
Node Manipulation	Nodes.Add(), Nodes.Remove(), Nodes.Insert(), Nodes.Clear()	Adds or removes a node to a Nodes collection. Insert adds a node at an indexed location, and Clear removes all tree nodes from the collection.
	Clone()	Copies a tree node and entire subtree.

Let's look at how TreeView and TreeNode members are used to perform fundamental TreeView operations.

Adding and Removing Nodes

The following code creates the tree in Figure 8.4 using a combination of Add, Insert, and Clone methods. The methods are applied on a pre-existing TreeView1 control.

```
TreeNode tNode;
// Add parent node to treeView1 control
tNode = treeView1.Nodes.Add("A");
// Add child node: two overloads available
tNode.Nodes.Add(new TreeNode("C"));
tNode.Nodes.Add("D");
// Insert node after C
tNode.Nodes.Insert(1, new TreeNode("E"));
// Add parent node to treeView1 control
tNode = treeView1.Nodes.Add("B");
tNode = treeView1.Nodes.Add("A");
tNode = treeView1.Nodes.Add("B");
tNode = treeView1.Nodes.Add("C");
tNode = treeView1.Nodes.Add("D");
tNode = treeView1.Nodes.Add("E");
tNode = treeView1.Nodes.Add("F");
tNode = treeView1.Nodes.Add("G");
tNode = treeView1.Nodes.Add("H");
tNode = treeView1.Nodes.Add("I");
tNode = treeView1.Nodes.Add("J");
tNode = treeView1.Nodes.Add("K");
tNode = treeView1.Nodes.Add("L");
tNode = treeView1.Nodes.Add("M");
tNode = treeView1.Nodes.Add("N");
tNode = treeView1.Nodes.Add("O");
tNode = treeView1.Nodes.Add("P");
tNode = treeView1.Nodes.Add("Q");
tNode = treeView1.Nodes.Add("R");
tNode = treeView1.Nodes.Add("S");
tNode = treeView1.Nodes.Add("T");
tNode = treeView1.Nodes.Add("U");
tNode = treeView1.Nodes.Add("V");
tNode = treeView1.Nodes.Add("W");
tNode = treeView1.Nodes.Add("X");
tNode = treeView1.Nodes.Add("Y");
tNode = treeView1.Nodes.Add("Z");
```

**Fig. 8.4** TreeView Node Representation

At this point, we still need to add a copy of node A and its subtree to the parent node B. This is done by cloning the A subtree and adding it to node B. Node A is referenced as TreeView1.Nodes[0] because it is the first node in the control's collection. Note that the Add method appends nodes to a collection, and they can be referenced by their zero-based position within the collection.

```
// Clone first parent node and add to node B
TreeNode      clNode      =      (TreeNode)
treeView1.Nodes[0].Clone();
tNode.Nodes.Add(clNode);
// Add and remove node for demonstration purposes
tNode.Nodes.Add("G");
tNode.Nodes.Remove(tNode.LastNode);
```

Iterating Through the Nodes in a TreeView

As with any collection, the foreach statement provides the easiest way to loop through the collection's members. The following code statements display all the top-level nodes in a control.

```
foreach (TreeNode tn in treeView1.Nodes)
{
    MessageBox.Show(tn.Text);
    // If (tn.Visible) true if node is visible
    // If (tn.Selected) true if node is currently
    selected
}
```

An alternate approach is to move through the collection using the TreeNode.NextNode property.

```
tNode = treeView1.Nodes[0];
while (tNode != null) {
    MessageBox.Show(tNode.Text);
    tNode = tNode.NextNode;
}
```

Detecting a Selected Node

When a node is selected, the TreeView control fires an AfterSelect event that passes a TreeViewEventArgs parameter to the event handling code. This parameter identifies the action causing the selection and the node selected. The TreeView example that follows illustrates how to handle this event. You can also handle the MouseDown event and detect the node using the GetNodeAt method that returns the node, if any at the current mouse coordinates.

```
private void treeView1_MouseDown(object sender,
MouseEventArgs e)
{
```

NOTES

```

TreeNode tn = treeView1.GetNodeAt(e.X, e.Y);
// You might want to remove the node: tn.Remove()
}

```

NOTES**A TreeView Example That Uses Reflection**

This example demonstrates how to create a simple object browser (refer to Figure 8.3) that uses a TreeView to display enumeration types for a specified assembly. When a node on the tree is clicked, the members for the selected enumeration are displayed in a ListView control.

Information about an assembly is stored in its metadata, and .NET provides classes in the System.Reflection namespace for exposing this metadata. The code in Example 8.1 iterates across the types in an assembly to build the TreeView. The parent nodes consist of unique namespace names, and the child nodes are the types contained in the namespaces. To include only enum types, a check is made to ensure that the type inherits from System.Enum.

Example 8.1: Using a TreeView and Reflection to List Enums in an Assembly

```

using System.Reflection;
//
private void GetEnums()
{
    TreeNode tNode=null;
    Assembly refAssembly ;
    Hashtable ht= new Hashtable(); // Keep track of
namespaces
    string assem = AssemName.Text; // Textbox with assembly
name
    tvEnum.Nodes.Clear();      // Remove all nodes from
tree
    // Load assembly to be probed
    refAssembly = Assembly.Load(assem);
    foreach (Type t in refAssembly.GetTypes())
    {
        // Get only types that inherit from System.Enum
        if (t.BaseType!=null      &&
t.BaseType.FullName=="System.Enum")
        {
            string myEnum = t.FullName;
            string nSpace =
                myEnum.Substring(0,myEnum.LastIndexOf("."));
            myEnum=
myEnum.Substring(myEnum.LastIndexOf(".") +1) ;
            // Determine if namespace in hashtable
        }
    }
}

```

```

    if( ht.Contains(nSpace) )
    {
        // Find parent node representing this namespace
        foreach (TreeNode tp in tvEnum.Nodes)
        {
            if(tp.Text == myEnum) { tNode=tp; break; }
        }
    }
    else
    {
        // Add parent node to display namespace
        tNode = tvEnum.Nodes.Add(nSpace);
        ht.Add(nSpace,nSpace);
    }
    // Add Child - name of enumeration
    TreeNode cNode = new TreeNode();
    cNode.Text= myEnum;
    cNode.Tag = t;   // Contains specific enumeration
    tNode.Nodes.Add(cNode);
}
}
}

```

NOTES

Observe how the reflection is used. The static Assembly.Load method is used to create an Assembly type. The Assembly.GetTypes is then used to return a Type array containing all types in the designated assembly.

```

refAssembly = Assembly.Load(assem);
foreach (Type t in refAssembly.GetTypes())

```

The Type.FullName property returns the name of the type which includes the namespace. This is used to extract the enum name and the namespace name. The Type is stored in the Tag field of the child nodes and is used later to retrieve the members of the enum.

After the TreeView is built, the final task is to display the field members of an enumeration when its node is clicked. This requires registering an event handler to be notified when an AfterSelect event occurs.

```

tvEnum.AfterSelect += new
    TreeViewEventHandler(tvEnum_AfterSelect);

```

The event handler identifies the selected node from the TreeViewEventArgs.Node property. It casts the node's Tag field to a Type class (an enumerator in this case) and uses the GetMembers method to retrieve the type's members as MemberInfo types. The name of each field member exposed by the MemberInfo.Name property is displayed in the ListView.

Tree and ListView

```

// ListView lView;
// lView.View = View.List;
private void tvEnum_AfterSelect(Object sender,
                                TreeViewEventArgs e)
{
    TreeNode tn = e.Node; // Node selected
    ListViewItem lvItem;
    if(tn.Parent !=null) // Exclude parent nodes
    {
        lView.Items.Clear(); // Clear ListView before adding
        items
        Type cNode = (Type) tn.Tag;
        // Use Reflection to iterate members in a Type
        foreach (MemberInfo mi in cNode.GetMembers())
        {
            if(mi.MemberType==MemberTypes.Field &&
               mi.Name != "value__" ) // skip this
            {
                lView.Items.Add(mi.Name);
            }
        }
    }
}

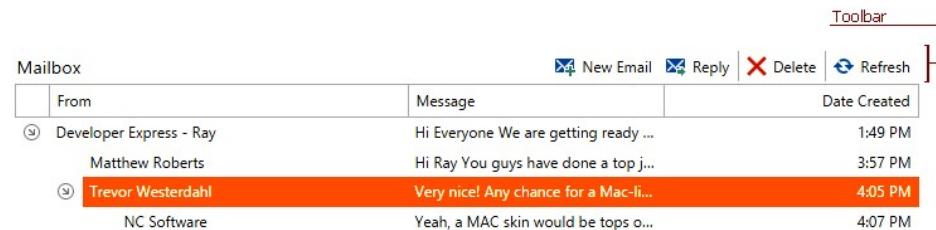
```

NOTES

8.2.1 Toolbar

A **Toolbar** can displays at the top and/or at the bottom of the ASPxTreeList. The screenshot given below shows an example of toolbar.

The table below lists the main members that affect element appearance and functionality.



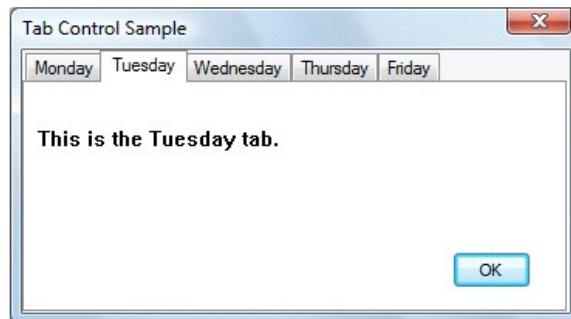
The screenshot shows a Windows application window titled 'Mailbox'. At the top, there is a toolbar with four buttons: 'New Email' (envelope icon), 'Reply' (reply icon), 'Delete' (trash icon), and 'Refresh' (refresh icon). Below the toolbar is a table representing an inbox. The table has columns: 'From', 'Message', and 'Date Created'. There are five rows of data:

From	Message	Date Created
Developer Express - Ray Matthew Roberts	Hi Everyone We are getting ready ... Hi Ray You guys have done a top j...	1:49 PM 3:57 PM
Trevor Westerdahl NC Software	Very nice! Any chance for a Mac-li... Yeah, a MAC skin would be tops o...	4:05 PM 4:07 PM

Tab Control

The Windows forms Tab Control displays multiple tabs, like dividers in a notebook or labels in a set of folders in a file cabinet. It splits the interface into different areas

each accessible by clicking on the tab header positioned at the top. Tab control allows to define multiple pages for the same area of window or dialog box where a certain type of information or a group of controls is displayed under the page of a particular tab. The following screenshot shows an application with tab control for the days of a week. The Tuesday tab is selected and the corresponding information is displayed.

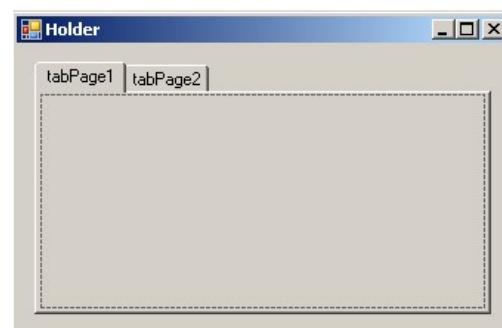


The tabs can contain pictures and other controls. You can use the tab control to produce the kind of multiple-page dialog box that appears many places in the Windows operating system, such as the Control Panel Display Properties. Additionally, the TabControl can be used to create property pages, which are used to set a group of related properties.

Working with Tab Control

The most important property of the TabControl is TabPages which contains the individual tabs. Each individual tab is a TabPage object. When a tab is clicked, it raises the Click event for that TabPage object. If only a tab control is created in the application, it will not play its intended role and is useless. To make it useful property pages must be added to the tab control.

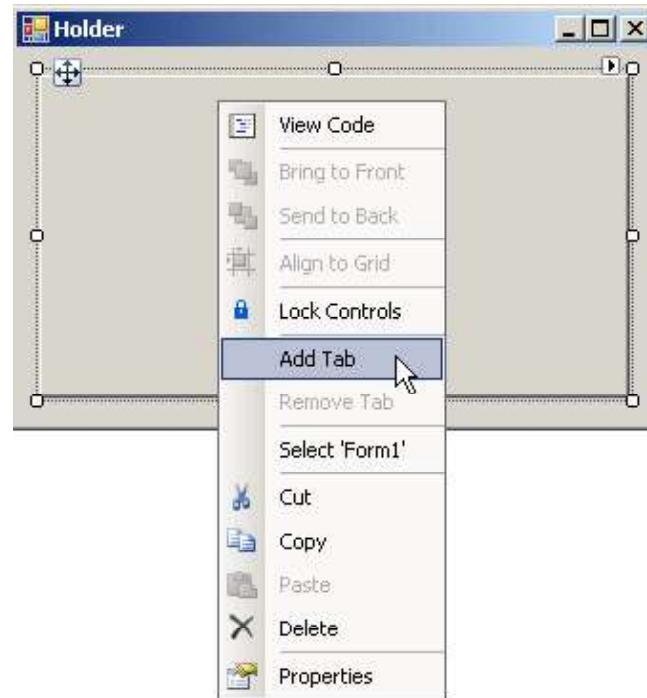
As mentioned earlier, the .Net framework provides the TabPage class to support property pages. TabPage class is not represented in the toolbox and must be added with the TabPage property of TabControl class. This property is of type TabPageCollection which in turn implements the IList, the ICollection, and the IEnumerable interface. The Tab control is created at the design time by dragging it from the toolbox and dropping it to the form. By default the tab control is equipped by two tabs as shown below.



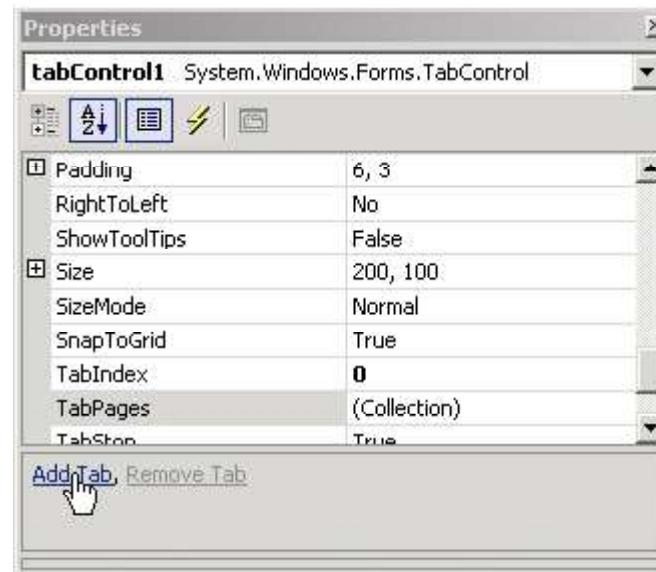
NOTES

If the tab control does not have any tabs, right click the tab control and click Add Tab option.

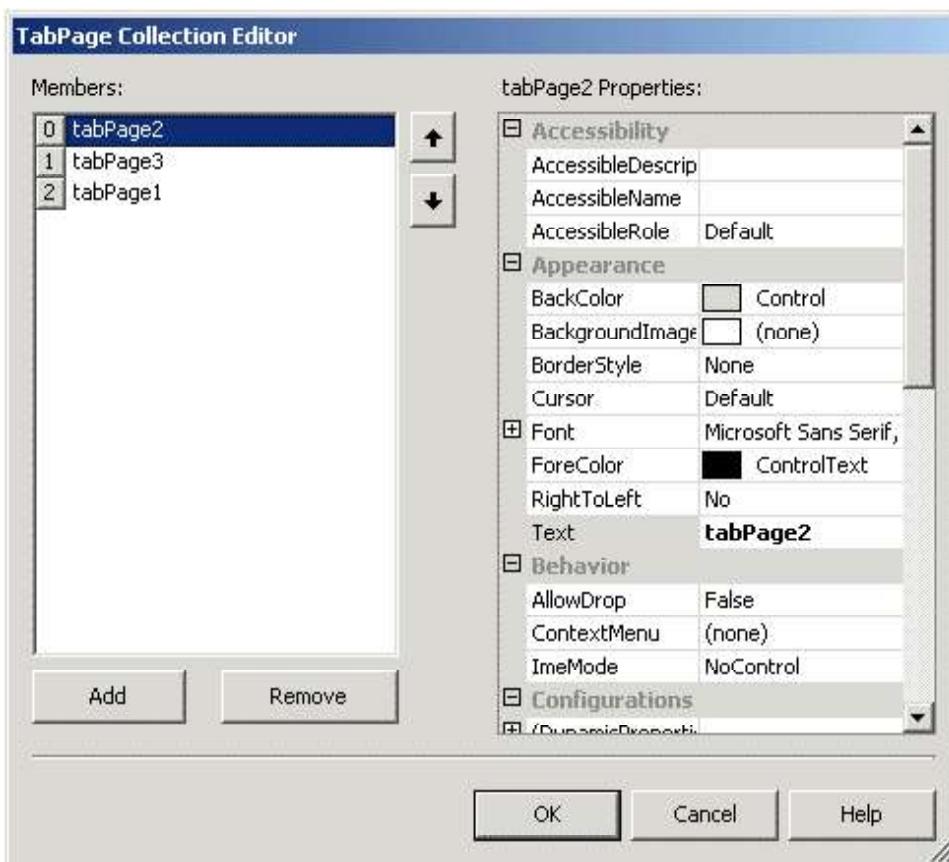
NOTES



Alternatively, the option can be found in the lower section of the properties window of the Tab control.



The tabs can also be added from the TabPages field to display theTabPage Collection Editor dialog box that allows user to create and configure each page by clicking its Add button.



Tree and ListView

NOTES

Similarly, to remove a tab Remove option can be clicked. SelectedTab or selectedIndex properties of the tab control can be used to select a tab page by default.

8.2.2 Status Bar Control

A status bar control displays information about an object being viewed in a window of an application, the object's component, or contextual information that relates to that object's operation within your application. It is used to provide information or some message such as data, time or about an action currently being performed in the application.

Status Bar can be created using a control StatusBar or StatusStrip (available in new versions) on the form. The following code shows how to add status bar in an application.

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

Public Class Exercise
    Inherits System.Windows.Forms.Form

    Dim statusbar As StatusStrip
```

```

    Public Sub New()
        statusbar = new StatusStrip
        Controls.Add(statusbar)
    End Sub

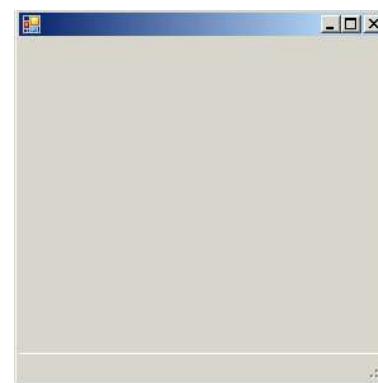
    Public Shared Function Main() As Integer
        Application.Run(New Exercise)
        Return 0
    End Function

    End Class

```

NOTES

The output of the above code will be:

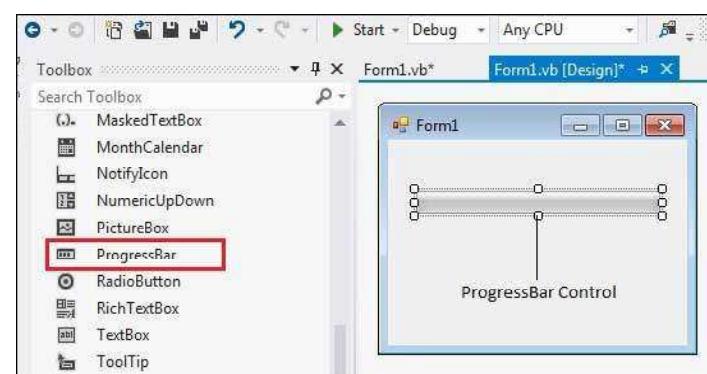


By default, the Dock value of status bar is Bottom. To create the panel, select the status bar and click the Edit Items option from the properties windows. This would open the Items Collection Editor to add the desired items to create the panels.

8.2.3 Progress Bar Control

It represents a Windows progress bar control. It is used to provide visual feedback to your users about the status of some task. It shows a bar that fills in from left to right with the operation progress.

Let's click on a ProgressBar control from the Toolbox and place it on the form.



The main properties of a progress bar are *Value*, *Maximum* and *Minimum*. The *Minimum* and *Maximum* properties are used to set the minimum and maximum values that the progress bar can display. The *Value* property specifies the current position of the progress bar.

The *ProgressBar* control is typically used when an application performs tasks such as copying files or printing documents. The application might look unresponsive to the user if there is no visual cue. In such cases, using the *ProgressBar* allows the programmer to provide a visual status of progress.

Table 8.2 Properties of the ProgressBar Control

S.No.	Property & Description
1.	AllowDrop Overrides <i>Control.AllowDrop</i> .
2.	BackgroundImage Gets or sets the background image for the <i>ProgressBar</i> control.
3.	BackgroundImageLayout Gets or sets the layout of the background image of the progress bar.
4.	CausesValidation Gets or sets a value indicating whether the control, when it receives focus, causes validation to be performed on any controls that require validation.
5.	Font Gets or sets the font of text in the <i>ProgressBar</i> .
6.	ImeMode Gets or sets the input method editor (IME) for the <i>ProgressBar</i> .
7.	ImeModeBase Gets or sets the IME mode of a control.
8.	MarqueeAnimationSpeed Gets or sets the time period, in milliseconds, that it takes the progress block to scroll across the progress bar.
9.	Maximum Gets or sets the maximum value of the range of the control.
10.	Minimum Gets or sets the minimum value of the range of the control.
11.	Padding Gets or sets the space between the edges of a <i>ProgressBar</i> control and its contents.
12.	RightToLeftLayout Gets or sets a value indicating whether the <i>ProgressBar</i> and any text it contains is displayed from right to left.
13.	Step Gets or sets the amount by which a call to the <i>PerformStep</i> method increases the current position of the progress bar.
14.	Style Gets or sets the manner in which progress should be indicated on the progress bar.
15.	Value Gets or sets the current position of the progress bar.

Table 8.3 Methods of the ProgressBar Control

S.No.	Method Name & Description
1.	Increment Increments the current position of the <i>ProgressBar</i> control by specified amount.
2.	PerformStep Increments the value by the specified step.
3.	ResetText Resets the <i>Text</i> property to its default value.
4.	ToString Returns a string that represents the progress bar control.

NOTES

Table 8.4 Events of the ProgressBar Control**NOTES**

S.No.	Event & Description
1.	BackgroundImageChanged Occurs when the value of the BackgroundImage property changes.
2.	BackgroundImageLayoutChanged Occurs when the value of the BackgroundImageLayout property changes.
3.	CausesValidationChanged Occurs when the value of the CausesValidation property changes.
4.	Click Occurs when the control is clicked.
5.	DoubleClick Occurs when the user double-clicks the control.
6.	Enter Occurs when focus enters the control.
7.	FontChanged Occurs when the value of the Font property changes.
8.	ImeModeChanged Occurs when the value of the ImeMode property changes.
9.	KeyDown Occurs when the user presses a key while the control has focus.
10.	KeyPress Occurs when the user presses a key while the control has focus.
11.	KeyUp Occurs when the user releases a key while the control has focus.
12.	Leave Occurs when focus leaves the ProgressBar control.
13.	MouseClick Occurs when the control is clicked by the mouse.
14.	MouseDoubleClick Occurs when the user double-clicks the control.
15.	PaddingChanged Occurs when the value of the Padding property changes.
16.	Paint Occurs when the ProgressBar is drawn.
17.	RightToLeftLayoutChanged Occurs when the RightToLeftLayout property changes.
18.	TabStopChanged Occurs when the TabStop property changes.
19.	TextChanged Occurs when the Text property changes.

Example 8.2: To create a progress bar at runtime.

Let's double click on the Form and put the following code in the opened window.

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) _
        Handles MyBase.Load
        'create two progress bars
        Dim ProgressBar1 As ProgressBar
        Dim ProgressBar2 As ProgressBar
        ProgressBar1 = New ProgressBar()
        ProgressBar2 = New ProgressBar()
        'set position
        ProgressBar1.Location = New Point(10, 10)
        ProgressBar2.Location = New Point(10, 50)
```

```

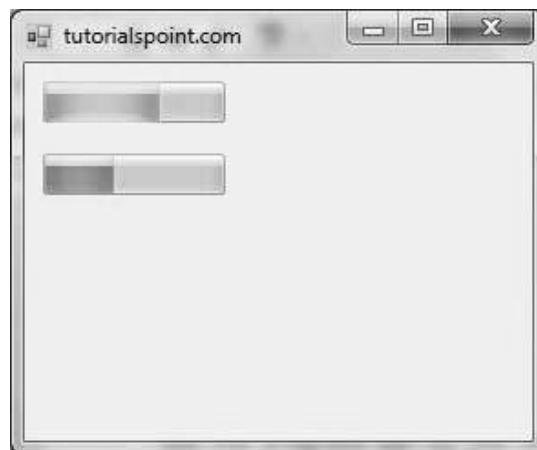
'set values
ProgressBar1.Minimum = 0
ProgressBar1.Maximum = 200
ProgressBar1.Value = 130
ProgressBar2.Minimum = 0
ProgressBar2.Maximum = 100
ProgressBar2.Value = 40
'add the progress bar to the form
Me.Controls.Add(ProgressBar1)
Me.Controls.Add(ProgressBar2)
' Set the caption bar text of the form.
Me.Text = "tutorialspoint.com"
End Sub
End Class

```

Tree and ListView

NOTES

When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window.



Check Your Progress

1. What is ListView?
2. What is the use of a status bar control?
3. What are the main properties of a progress bar?

8.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. ListView is another window control that displays textual information as a collection of items in rows and columns with column heading. It allows user to create an interface like Window Explorer.

NOTES

2. A status bar control displays information about an object being viewed in a window of an application, the object's component, or contextual information that relates to that object's operation within your application.
3. The main properties of a progress bar are *Value*, *Maximum* and *Minimum*.

8.4 SUMMARY

- ListView is another window control that displays textual information as a collection of items in rows and columns with column heading. It allows user to create an interface like Window Explorer.
- In ListView, View property can be changed at runtime to switch among the possible views. In fact, you can recognize that the view options correspond exactly to the view menu options available in Windows Explorer.
- Several overloaded forms of the ListView constructor are available. They can be used to create a single item or a single item and its sub items.
- Two collections of images can be associated with a ListView control as ImageList properties. LargeImageList contains images used in the LargeIcon view; and SmallImageList contains images used in all other views.
- Common tasks associated with the ListView control include iterating over the contents of the control, iterating over selected items only, detecting the item that has focus and sorting the items by any column.
- TreeView control provides a tree-like view of hierarchical data as its user interface. Underneath, its programming model is based on the familiar tree structure consisting of parent nodes and child nodes.
- The Windows forms Tab Control displays multiple tabs, like dividers in a notebook or labels in a set of folders in a file cabinet. It splits the interface into different areas each accessible by clicking on the tab header positioned at the top. Tab control allows to define multiple pages for the same area of window or dialog box where a certain type of information or a group of controls is displayed under the page of a particular tab.
- A status bar control displays information about an object being viewed in a window of an application, the object's component, or contextual information that relates to that object's operation within your application.
- Progress bar control is used to provide visual feedback to your users about the status of some task. It shows a bar that fills in from left to right with the operation progress.

8.5 KEY WORDS

- **TreeView Control:** It is a graphical control element that presents a hierarchical view of information.

- **Status Bar Control:** It displays information about an object being viewed in a window of an application, the object's component, or contextual information that relates to that object's operation within your application.
- **ProgressBar Control:** It is used to provide visual feedback to your users about the status of some task.

NOTES

8.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Discuss the various properties of ListView.
2. How you will create the ListView Items?
3. Write a note on TreeView Class.
4. Discuss the process of creating the tab control.

Long Answer Questions

1. Explain the various tasks associated with ListView control.
2. How you will add and remove the nodes in the TreeView control?
3. Write the code that shows how to add status bar in an application.
4. Explain some of the properties, methods and events of the ProgressBar control.

8.7 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis. 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 9 DEBUGGING AND ERROR HANDLING

NOTES

Structure

- 9.0 Introduction
 - 9.1 Objectives
 - 9.2 Errors
 - 9.2.1 Error Checking Versus Exception Handling
 - 9.2.2 Types of Errors
 - 9.2.3 Error Handling
 - 9.3 Exception Handling
 - 9.3.1 Structured Exception Handling
 - 9.3.2 Finally Block
 - 9.3.3 Unstructured Exception Handling on Error Statement
 - 9.4 Answers to Check Your Progress Questions
 - 9.5 Summary
 - 9.6 Key Words
 - 9.7 Self Assessment Questions and Exercises
 - 9.8 Further Readings
-

9.0 INTRODUCTION

Exception handling is a programming language construct or computer hardware mechanism designed to handle the occurrence of exceptions, special conditions that change the normal flow of program execution. In this unit, you will be familiarized with exception handling and why it is basically useful. Exception handling is required to prevent errors from occurring. The unit will differentiate between error checking and exception handling. Apart from exception handling, you will also learn about error handling. Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications. The unit will also introduce you to the `try...catch` block and you will learn how to analyse an exception. Finally, the unit will discuss throwing exception.

9.1 OBJECTIVES

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

- Learn about exception handling
- Differentiate between error checking and exception handling
- Learn about error handling
- Learn about the `try...catch` block and analysing the exception
- Understand throwing exceptions

9.2 ERRORS

NOTES

To prevent errors from occurring after you have distributed your application, you need to implement error trapping and handling. This will require you to write a good error handler that anticipates problems or conditions that are not under the control of your application and that will cause your program to execute incorrectly at runtime. You can achieve this mainly during the planning and design phase of the application. This needs a deep understanding of how your application should work, and the anomalies that may pop up at runtime.

For runtime errors that occur because of conditions that are above a program's control, you handle them by using exception handling and checking for common problems before executing an action. An instance of checking errors would be to make sure that a floppy disk is in the drive before trying to write to it or to make sure that a file exists before trying to read from it. Another instance of when to use exception handling is retrieving a record set from a database. You might have a valid connection, but something might have happened to your connection which caused retrieval of a record set to fail. You could use exception handling to trap this error rather than having a cryptic message popping up and then aborting your application.

You must use exception handling to prevent your application from aborting. They accommodate support for handling runtime errors, called exceptions, which can occur when your program is running. Using exception handling, a program can take steps to recover from abnormal events outside your program's control rather than crashing your application. These exceptions are handled by code that is not run during normal execution.

In previous versions of Visual Basic, error handling used the `On Error Goto` statement. One of the complaints fabricated by programmers has been the lack of exception handling. Visual Basic.NET meets this requirement with the inclusion of the `Try...Catch...Finally` exception handling. Those of you who have programmed in C++ should already be familiar with this concept.

9.2.1 Error Checking Versus Exception Handling

Error checking in the earlier version of the VB was done by returning the value of a given function and reacting based in that value. This usually used the switch statement that checks the value returned by the function. The return value normally tends to be random and the result comes in the Boolean format; if the function returns 1, it means result is bad otherwise a 0 can mean the success or just often failure. To understand this, look at the following example:

```
Select Case ErrorNumber  
Case 57  
    MsgBox "Your printer may be off-line."  
Case 68
```

NOTES

```
MsgBox "Is there a printer available?"  
'more cases...  
Case Else  
'eeks  
End Select
```

While talking about the exception handling, you must keep in mind some things that are useful when applying structure exception handling. You can ignore the rules which are applicable at the time the exception occur. When you apply the structure of the exception handling you must apply the path that will execute automatically when the exception occurs. You can create an alternate path for code execution when the code cannot complete its work in the normal path. Also, when you enable exception handling, VB.NET automatically creates an object that encapsulates the error information.

When any exception occurs in the program, the built in mechanism of the exception handing searches the object which handles the exception. Exception handing gives a way to design you own error or message in the application when application is in the use. You first mention all the problems that can occur in the code at the time of execution and at same time you have the solution of the problem. In VB.NET, the actual mechanism of the exception handling is using Try Catch Block.

Analysing the Exception

This is done by modifying the catch statement to read something as the following:

```
Catch exe as exception
```

You can use any variable here because it is being declared in the catch statement clause. Now the exception object referenced by the exe contains a lot of information.

Change the code in the code in the catch clause to read the following:

```
Catch exe as exception  
Console.WriteLine(exe)
```

If you use the advantage of the built in in the ToString method of the exception object exe, you will have the following error:

```
System.IndexOutOfRangeException: An exception of type_ _  
System.IndexOutOfRangeException was thrown at  
Exception_1.Exception1.Main() in C:\Documents  
and_Settings\x20\My Documents\Visual Studio  
Projects\ConsoleApplication14\Exception1.vb:line 6
```

This message shows there was an error in accessing the array element at line 6. You can make the catch clause throw an exception object back to the code that called it that encapsulate what went wrong in its cleanup work.

9.2.2 Types of Errors

No matter how hard we try, errors are caused within our programs. These errors can be grouped into three categories:

- (i) Logic errors
- (ii) Runtime errors
- (iii) Syntax errors

NOTES

(i) Logic errors

They are the hardest to find. With logic errors, the program usually runs, but produces incorrect or unexpected results. The Visual Basic debugger aids in detecting logic errors.

(ii) Runtime errors

They are usually beyond your program's control. They occur when a variable takes on an unexpected value (divided by zero) or when a drive door is left open or when a file is not found. Visual Basic allows you to trap such errors and make attempts to correct them.

(iii) Syntax errors

They occur when a command is mistyped or an expected phrase or argument is left out. Visual Basic detects these errors as they occur and even provides help in correcting them. You cannot run a Visual Basic program until all syntax errors have been corrected.

Methods to minimize errors

Errors can be minimized in the following ways:

- Design your application carefully. More design time means less debugging time.
- Use comments where applicable to help you remember what you were trying to do.
- Use consistent and meaningful naming conventions for your variables, objects and procedures.

9.2.3 Error Handling

First you need to understand what an error is in error handling. Errors are compile time or run time mistakes that a programmer makes. Errors are managed by the compiler by giving a message regarding any mistake; error handling is basically designed to make programming simpler. Error handling is used to increase the programming productivity. There are the following two types of errors:

- Syntax error
- Runtime error

NOTES

Syntax errors are those errors which are handled at compile time. Compiler checks all the codes and finds errors in those codes, which were made by the user at design time. This type of mistake is done at compile time. Till the compiler find the code is in the right form, it never executes that code.

Runtime errors are those errors which are handled at run time. These types of errors find errors like wrong out, unexpected result, etc. Programmers with good knowledge know they live in a world where exceptions occur often

VB.NET also supports structured exception handling for handling the common errors. You will learn how exception handling is used and the syntax used to add exception handling to a VB.NET application. First of all you need to know about the benefits of using structured exception handling for error handling.

In earlier versions of VB, errors were checked by checking the return value of a given function. This was usually done by using the equivalent of a switch statement that checks the value returned by the function and this return resulted in the form of 1 and 0. 1 is good sometimes and bad other times; a 0 mean success or just as often failure. In the following example code, the value returned seems truly random:

```
Select Case ErrorChecknum
Case 57
    MsgBox "undefined value "
Case 68
    MsgBox "Is there a syntax error?"
'more cases- Case Else
'veeks
End Select
```

This type of work is tough to handle and that is why you need a convenient way through which you can work faster and increase the processing speed. So to finish this type of problem exception handling is used.

Before you consider writing the code that shows you some exception handlers at work, some points need to be kept in mind. First, when you use structured exception handling, you are providing an alternative path for your code that will be executed automatically when something bad happens. Second, and more importantly, you can create in any VB.NET code an alternate path for code execution when the code cannot complete its work in the normal path. Also, when you enable exception handling, VB.NET automatically creates an object that encapsulates the error information.

VB.NET provides three keywords `try`, `catch` and `finally` to do exception handling. The `try` encloses the statements that might throw an exception whereas `catch` handles an exception if one exists. The `finally` can be used for doing any clean up process.

The general form of try...catch...finally in VB.NET is as follows:

```
Try' Statement which can cause an exception.  
Catch x As Type'  
Statements for handling the exception  
Finally  
End Try 'Any cleanup code
```

NOTES

If any exception occurs inside the `try` block, the control transfers to the appropriate `catch` block and later to the `finally` block. However in VB.NET, both `catch` and `finally` blocks are optional. The `try` block can exist either with one or more `catch` blocks or a `finally` block or with both `catch` and `finally` blocks. If no exception occurred inside the `try` block, the control directly transfers to `finally` block. You can say that the statements inside the `finally` block is always executed.

Note: It will give you an error if you transfer control out of a `finally` block by using `break`, `continue`, `return` or `goto`.

In VB.NET, exceptions are nothing but objects of the type `Exception`. The `Exception` is the ultimate base class for any exceptions in VB.NET. VB.NET itself provides some standard exceptions; or even the user can create their own exception classes, provided that this should inherit from either `Exception` class or one of the standard derived classes of `Exception` class like `DivideByZeroException` to `ArgumentException`, etc.

Check Your Progress

1. What is an error?
2. List the two types of errors.
3. What are the uses of `try`, `catch` and `finally` keywords?

9.3 EXCEPTION HANDLING

The terms, error and exception, are often used interchangeably. In fact, an error, which is an event that happens during the execution of code, interrupts or disrupts the code's normal flow and creates an exception object. When an error interrupts the flow, the program tries to find an exception handler — a block of code that tells it how to react — that will help it resume the flow. In other words, an error is the event; an exception is the object that the event creates.

Programmers use the phrase “throwing an exception” to mean that the method in question encountered an error and reacted by creating an exception object that contains information about the error and when/where it occurred. Factors that cause errors and subsequent exceptions include user error, resource failures, and failures of programming logic. Such errors are related to how the code undertakes a specific task; they are not related to the purpose of the task.

NOTES

For the purpose of this article, “exception handling” means interpreting and reacting to the exceptions created by errors.

Structured exception handling is simply that — using a control structure containing exceptions, isolated blocks of code, and filters to create an exception handling mechanism. This allows your code to differentiate between different types of errors and react in accordance with circumstances. In unstructured exception handling, an **On Error** statement at the beginning of the code handles all exceptions.

Structured exception handling is significantly more versatile, robust, and flexible than unstructured. If possible, use structured exception handling. However, you might use unstructured exception handling under these circumstances:

- You are upgrading an application written in an earlier version of Visual Basic.
- You are developing a preliminary or draft version of an application and you don’t mind if the program fails to shut down gracefully.
- You know in advance exactly what will cause the exception.
- A deadline is pressing and you need to take shortcuts.
- Code is trivial or so short that you only need to test the branch of code generating the exception.
- You need to use the **Resume Next** statement, which is not supported in structured exception handling.

You cannot combine structured and unstructured exception handling in the same function. If you use an **On Error** statement, you cannot use a **Try...Catch** statement in the same function.

Regardless of which you choose to handle exceptions within your code, you must take a step back and examine what assumptions that code makes. For example, when your application asks the user to input a telephone number, the following assumptions come into play:

- The user will input a number rather than characters.
- The number will have a certain format.
- The user will not input a null string.
- The user has a single telephone number.

User input might violate any or all of these assumptions. Robust code requires adequate exception handling, which allows your application to recover gracefully from such a violation.

Unless you can guarantee that a method will never throw an exception under any circumstances, allow for informative exception handling. Exception handling should be meaningful. Beyond stating that something went wrong, messages resulting from exception handling should indicate why and where it went wrong. An

uninformative message along the lines of “An error has occurred” only frustrates the user.

Debugging and Error handling

9.3.1 Structured Exception Handling

Try...catch block provides the actual mechanism that helps in handling the exception in VB.NET. The actual way for doing this in VB .NET is called a block. For instance, suppose you build a console application in which the user is supposed to use the application from the command line by typing the following syntax of try...catch block:

```
Try
    'code that created a local object that has a Dispose
    method
    ' more code that might throw exceptions
Catch(e As Exception)
    localObject.dispose()
    Throw e;
End Try
```

Consider the following example:

```
Module Exception1
    Sub Main()
        Dim args() As String
        Try
            args = Environment.GetCommandLineArgs()
            ProcessFile(args(1))
        Catch
            Console.WriteLine("ERROR")
        End Try
        Console.WriteLine("Press enter to end")
        Console.ReadLine()
    End Sub
    Sub ProcessFile(ByVal fileName As String)
        'process file code goes here
        Console.WriteLine("Am processing " & fName)
    End Sub
End Module
```

A process file is called when any type of error occurs in the try...catch block. In this code snippet, if the user does not enter a filename, then the code would try to catch exception name `IndexOutOfRangeException` of the file. Triggering this exception causes the Catch code flow to move down the alternate pathway, which in this ERROR Case simply prints out in the console window.

NOTES

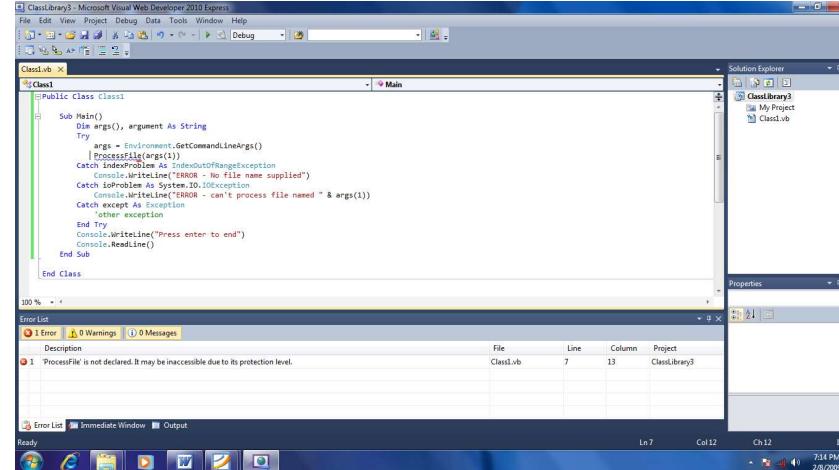
NOTES

Multiple Catch Clauses

The VB.NET runtime allows multiple clauses and each type of clause can trap a specific Exception using objects that inherit from the base class to identify particular errors. For example, consider the following code:

```
Public Class Class1
    Sub Main()
        Dim args(), argument As String
        Try
            args = Environment.GetCommandLineArgs()
            ProcessFile(args(1))
        Catch indexProblem As IndexOutOfRangeException
            Console.WriteLine("ERROR - No file name supplied")
        Catch ioProblem As System.IO.IOException
            Console.WriteLine("ERROR - can't process file named
" & args(1))
        Catch except As Exception
            'other exception
        End Try
        Console.WriteLine("Press enter to end")
        Console.ReadLine()
    End Sub
End Class
```

The following screenshot shows how the program will appear on screen.



The exception handler block attempts to match all of the blocks sequentially to find a match. If the user leaves out the filename, it will match the first clause. Probably it will match the second clause `ProcessFile` if the call cannot process the file.

There is another example in by which you can see multiple catch clauses. Consider the following example:

```
Try
    ProcessFile(args(1))
    Catch indexProblem As IndexOutOfRangeException
        Console.WriteLine("ERROR - No file name supplied")
    Catch ioProblem As System.IO.IOException
        Console.WriteLine("ERROR - can't process file named "
& args(1))
    Catch fileNotFound As System.IO.FileNotFoundException
    End Try
```

NOTES

In this example, three types of exceptions are considered. They are as follows:

- (i) `IndexOutOfRangeException`
- (ii) `System.IO.IOException`
- (iii) `System.IO.FileNotFoundException`

It was considered that the exception handling method would simply return the exception back. In this procedure, Try containing a block handles the errors, so the exception handling that was written should handle it.

Throw

A method is capable of causing an exception that does not handle. Throw clause lists the type of exception that a method might throw. This is necessary for all exception except throw of type error or runtime Exception or any other subclasses. All other exceptions that a method can throw must be declared in the throw clause. Following is an example to understand the code using throw:

```
Imports System
Class MyClient
    Public Shared Sub Main()
        Try
            Throw New DivideByZeroException("Invalid Division")
        Catch e As DivideByZeroException
            Console.WriteLine("Exception")
        End Try
        Console.WriteLine("LAST STATEMENT")
    End Sub 'Main
End Class 'MyClient
Finally
```

When exception is thrown, execution in a method takes a rather abrupt and non-linear way to alternate the normal flow through the method depending upon how the method code is coded or written.

NOTES

Re-Throwing Exception

The exceptions, which you caught inside a `catch` block, can be re-thrown to a higher context using the keyword `throw` inside the `catch` block. The following program shows how to do this:

```
Imports System
Class MyClass
    Public Sub Method()
        Try
            Dim y As Integer = 0
            Dim Sum As Integer = 100 / y
            Catch e As DivideByZeroException
                Throw
            End Try
        End Sub 'Method
    End Class
    Class MyUser
        Public Shared Sub Main()
            Dim zz As New MyClass
            Try
                zz.Method()
                Catch e As Exception
                    Console.WriteLine("Exception caught here")
                End Try
                Console.WriteLine("LAST STATEMENT")
            End Sub
        End Class
    
```

Standard Exceptions

There are two types of exceptions, viz., exceptions generated by an executing program and exceptions generated by the common language runtime. `System.Exception` is the base class for all exceptions in VB.NET. Several exception classes inherit from this class including `ApplicationException` and `SystemException`. These two classes form the basis for most other runtime exceptions. Other exceptions that derive directly from `System.Exception` include `IOException`, `WebException`, etc.

The common language runtime throws `SystemException`. A user program rather than the runtime throws the `ApplicationException`.

The `SystemException` includes the `ExecutionEngineException`, `StackOverflowException`, etc. It is not recommended that you catch

SystemExceptions, nor is it good programming practice to throw SystemExceptions in your applications. Consider the following:

```
System.OutOfMemoryException  
System.NullReferenceException  
Syste.InvalidCastException  
Syste.ArrayTypeMismatchException  
System.IndexOutOfRangeException  
System.ArithemeticException  
System.DevideByZeroException  
System.OverFlowException
```

NOTES

User-Defined Exceptions

In VB.NET, it is possible to create your own exception class. However, Exception must be the ultimate base class for all exceptions in VB.NET. So the user-defined exception classes must inherit from either Exception class or one of its standard derived classes. Consider the following example:

```
Imports System  
Class MyException  
Inherits Exception  
Public Sub New(ByVal ptr As String)  
Console.WriteLine("User defined exception")  
End Sub 'New  
End Class 'MyException  
Class MyUser  
Public Shared Sub Main()  
Try  
Throw New MyException("Radix")  
Catch err As Exception  
Console.WriteLine(("Exception caught here" +  
err.ToString()))  
End Try  
Console.WriteLine("LAST STATEMENT")  
End Sub  
End Class
```

A new exception class was created that was inherited from IOException, because problems were occurring. Sometimes it happens that there is no obvious class to inherit from except Exception itself. In such cases, it is suggested not to inherit from Exception itself, but rather a subclass of Exception called ApplicationException is used. The reason is that the .NET Framework distinguishes between exceptions that arise due to problems caused by the runtime

NOTES

(such as running out of memory or stack space) and those caused by your application. It is the latter exceptions that are supposed to inherit from `ApplicationException`, and therefore this is the class you should inherit from when you create a generic exception in your program.

The runtime tries to help you by going a little further. It actually splits the exception hierarchy into two.

The `Exception`, `ApplicationException`, and `SystemException` classes have identical functionality—the existence of the three classes is a convenience that makes the exceptions in your programs easier to understand.

By combining exception handling and building your own exception classes, you can finally eliminate all uses of the `GoTo`. It is more likely to just wrap the whole loop in a `Try...Catch` block as follows:

```
Sub Main()
    Dim getData As String
    Dim i, j As Integer
    Dim e As System.IO.IOException
    Try
        For i = 1 To 10
            For j = 1 To 100
                Console.WriteLine("Type the data, hit the Enter key between "
                    & _
                    "ZZZ to end: ")
                getData = Console.ReadLine()
                If getData = "ZZZ" Then
                    e = New System.IO.IOException("Data entry ended at
user request")
                    Throw e
                Else
                    'Process data
                End If
            Next j
        Next i
        Catch
            Console.WriteLine(e.Message)
            Console.ReadLine()
        End Try
    End Sub
```

9.3.2 Finally Block

As the name suggests, `finally` means a final statement in the application. Imagine when you use the `try...catch` block, there is some clean up code that must be executed normally. Files should be closed in all condition. In the example discussed

later, `Display` method need to be called, and so on, but you can assure that certain codes are executed no matter what happen by adding a `finally` clause as in the example.

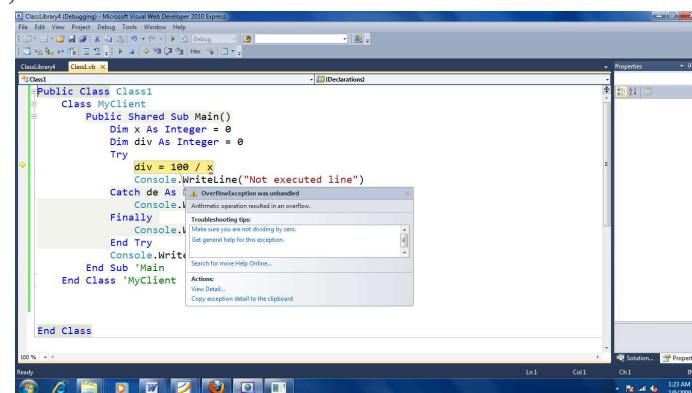
When exceptions are thrown, execution in a method is rather abrupt and non-linear that changes the normal flow through the method. It depends on how the method code is coded or written.

`Finally` creates a block of code that will be executed right after `try...catch` block has completed and before the code following the `try...catch` block executes. The `finally` block will execute whether or not an exception is thrown. If an exception is thrown, the `finally` block will execute even if `try...catch` statement catch the exception. A method can anytime return a call from inside a `try...catch` block. This can be useful for closing a file handle and freeing up any resources that might have been allocated at the beginning of a method with the intent of exposing them before running.

The `finally` clause is optional, however `try...catch` block requires at least one `catch` and one `finally` clause. Following is the example of exception handling using `finally` class:

```
Imports System
Class MyClient
    Public Shared Sub Main()
        Dim x As Integer = 0
        Dim div As Integer = 0
        Try
            div = 100 / x
            Console.WriteLine("Not executed line")
        Catch de As DivideByZeroException
            Console.WriteLine("Exception occurred")
        Finally
            Console.WriteLine("Finally Block")
        End Try
        Console.WriteLine("Result is {0}", div)
    End Sub 'Main
End Class 'MyClient
```

When you apply the program just mentioned, the following error, shown on the screenshot, will occur.



NOTES

NOTES

9.3.3 Unstructured Exception Handling

Visual Basic has specific built-in ways to handle run-time errors called trappable errors. Trapping errors is an indication to the operating system that you will handle the error. When such an error occurs, you can direct the execution of your program to an error handler, which is a section of code written specifically to deal with errors.

VB provides a set of `On Error` statements, `Err` and `Error` objects that you can use to trap errors.

On Error Statement

An `On Error` statement is used by VB for registering the error-handling code. This statement can take one of the following forms:

- `On Error GoTo 0`
- `On Error GoTo Label`
- `On Error Resume Next`
- `On Error GoTo line`

These are the forms by which VB finds out what it should do when an error is encountered by a program. Let us now discuss these four forms of the `On Error` statements one by one.

Using On Error GoTo 0

`On Error GoTo 0` is rather straightforward. Basically, it disables any enabled error handler in the current procedure. In other words, it cancels the currently installed error handler assigned by the previous `On Error GoTo Line` statement or the `On Error Resume Next` statement. The program crashes if it encounters any further errors.

Using On Error GoTo Label

The Visual Basic `On Error GoTo` statement is the foundation of handling trappable errors. When you execute an `On Error GoTo Label` statement in your code, execution is transferred to the code starting at `Label` if a trappable error has occurred. The code following that label is your error handler.

Using On Error Resume Next

The `On Error Resume Next` statement provides an easy way to disregard errors, if you want to do so. Once you execute this statement, execution continues with the next line of code if the current line generates an error and the error is ignored. While using `On Error Resume Next` statement, the program must check the `Err` object subsequent to each operation that might lead to an error. An `Err` object is a special VB object that contains information about run-

time errors. It has a special property called `Number` that sets or returns a numeric value that specifies an error.

If the `On Error` statement is not used, then the run-time errors that might occur may prove to be fatal; in other words, an error message will be displayed and execution will stop.

If the value `Err.Number` turns out to be non-zero, then the operation causing the error can be discovered and special action can be taken by the program. `Err.Number` should be checked by the program immediately after the statement is generated as some other action might reset the `Err` object and take away the prior error information.

Using On Error GoTo Line

A new error handler is registered by the `On Error GoTo` line statement. If an error is encountered by a program, the control is passed by it to the error handler starting at the indicated line label or number. The error handler or the error-handling routine can then take appropriate action. An error-handling routine is not a `Function` or `Sub` procedure. It is a section of code marked by a line number or label. Numbering code lines is a part of the Visual Basic history that goes all the way back to the original days of the Basic language, and in fact, many programmers do not know that they can number the lines of code in Visual Basic.

A simple error handler that catches unexpected errors and describes them to the user is illustrated in the following code:

```
Private Sub Dothings()
    ' Installing the error handler. See the code line below
    On Error GoTo UnexpectedError
        ' Do some stuff.
        ' Do not pass through into the error
    handler code.

    Exit Sub
UnexpectedError:
    ' Describe the error to the user
    MsgBox "Unexpected error" + Str$(Err.Number) + "in
    subroutine Dothings." + vbCrLf + -
    Err.Description

    Exit Sub
End Sub
```

The constant `vbCrLf` can be used for inserting carriage return/linefeed combinations.

NOTES

NOTES

Nested error handling

If a trappable error occurs in a procedure, you can handle that error through an error handler. But what if you call another procedure and an error occurs before control returns from that procedure? If the called procedure has an error handler, the code in that error handler will be executed. However, if the called procedure does not have an error handler, then control will return to the error handler in the calling procedure. In this way, control moves back up the calling chain to the closest error handler.

Err object

You need to make your error-handling code appropriate for the type of error that triggers it. VB supplies the `Err` object for this with a number of useful properties and methods. The `Number` property is the only one you are likely to need.

Check Your Progress

4. When is a process file called?
5. What are the two types of exceptions?

9.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Errors are compile time or run time mistakes that a programmer makes.
2. The two types of errors are syntax errors and runtime errors.
3. The `try` encloses the statements that might throw an exception whereas `catch` handles an exception if one exists. The `finally` can be used for doing any clean up process.
4. A process file is called when any type of error occurs in `try...catch` block.
5. The two types of exceptions are exceptions generated by an executing program and exceptions generated by the common language runtime.

9.5 SUMMARY

- To prevent errors from occurring after you have distributed your application, you need to implement error trapping and handling.
- For runtime errors that occur because of conditions that are above a program's control, you handle them by using exception handling and checking for common problems before executing an action.
- Error checking in the earlier version of the VB was done by returning the value of a given function and reacting based in that value.

- You can ignore the rules which are applicable at the time the exception occur. When you apply the structure of the exception handling you must apply the path that will execute automatically when the exception occurs.
- When any exception occurs in the program, the built in mechanism of the exception handing searches the object which handles the exception.
- The two types of errors are syntax errors and runtime errors.
- VB.NET provides three keywords `try`, `catch` and `finally` to do exception handling.
- If any exception occurs inside the `try` block, the control transfers to the appropriate `catch` block and later to the `finally` block.
- Try...catch block provides the actual mechanism that helps in handling the exception in VB.NET.
- A process file is called when any type of error occurs in the try...catch block.
- The VB.NET runtime allows multiple clauses and each type of clause can trap a specific `Exception` using objects that inherit from the base class to identify particular errors.
- The exception handler block attempts to match all of the blocks sequentially to find a match.
- A method is capable of causing an exception that does not handle. `Throw` clause lists the type of exception that a method might throw.
- The exceptions, which you caught inside a `catch` block, can be re-thrown to a higher context using the keyword `throw` inside the `catch` block.
- There are two types of exceptions, viz., exceptions generated by an executing program and exceptions generated by the common language runtime.
- The `Exception`, `ApplicationException`, and `SystemException` classes have identical functionality—the existence of the three classes is a convenience that makes the exceptions in your programs easier to understand.
- `Finally` creates a block of code that will be executed right after `try...catch` block has completed and before the code following the `try...catch` block executes.
- The `finally` clause is optional, however `try...catch` block requires at least one `catch` and one `finally` clause.

NOTES

9.6 KEY WORDS

- **Exceptions:** Refers to runtime errors, which can occur when the program is running
- **Syntax Errors:** Errors handled at compile time

9.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short Answer Questions

1. What is exception handling? Why is it used? What happens when an exception occurs in the program?
2. What important points are to be considered before writing code that shows exception handlers at work?
3. How does the `try...catch...finally` block function?
4. Write a note on multiple catch clauses.
5. What are standard exceptions?
6. What happens when there is no class to inherit from except `Exception` class?

Long Answer Questions

1. What is error handling? Describe the two types of errors.
2. Elaborate throwing exceptions in detail.
3. Write a detailed note on user-defined exceptions.
4. Elaborate upon ...Finally block.

9.8 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

BLOCK - IV

ASP.NET

NOTES

UNIT 10 INTRODUCTION TO ASP.NET

Structure

- 10.0 Introduction
- 10.1 Objectives
- 10.2 File Types in ASP.NET
- 10.3 Page Class
- 10.4 HttpRequest
- 10.5 HttpResponse Class
- 10.6 Server Utility
- 10.7 Answers to Check Your Progress Questions
- 10.8 Summary
- 10.9 Key Words
- 10.10 Self Assessment Questions and Exercises
- 10.11 Further Readings

10.0 INTRODUCTION

Active Server Pages (ASP) is a file extension for the file format used by an HTML file that contains a script processed by Microsoft server and is designed for ASP.Net framework. It provides a programming model along with a comprehensive software infrastructure and other services which are required to develop a robust web applications for PCs and mobile devices. ASP.Net works on the top of the HTTP protocol, which is a standard protocol used across all web application. It uses HTTP commands and policies to establish a bilateral communication between browser (client) and server. The basic architecture of the ASP.Net framework is based on the following components.

- 1. Language:** There are variety of languages available to write the ASP.Net applications code such as C#, VB.Net, Jscript, J#. Any of the language can be used to develop web applications.
- 2. Library:** There are various standard class libraries of reusable types (classes, interfaces, structures, and enumerated values are collectively called types) are included in .Net framework. The most commonly used is the Web library to develop a web application, which has all the required components to develop .Net based applications.
- 3. Common Language Runtime (CLR):** CLR is responsible for managing various key activities such as memory management, exception handling,

NOTES

security checking, garbage collection, debugging, thread execution, code execution, code safety, verification and compilation. The code which is directly managed by CLR is called the managed code which when compiled is converted into a CPU independent intermediate language (IL) code by the compiler. The IL code when compiled into native code by the Just in Time (JIT) compiler, is a CPU specific code.

ASP.Net is a part of Microsoft .Net platform used to produce interactive, data driven web application on the Internet.

10.1 OBJECTIVES

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

- Explain the different types of files in ASP.NET
- Understand how to import namespaces
- Discuss the usage of Global.asax file
- Understand the significance of page class
- Understand the use of HttpRequest class
- Understand the use of HttpResponse class

10.2 FILE TYPES IN ASP.NET

ASP.NET development uses specific file types depending on the resource used. The following list provides various types of files that you may use during ASP.NET development.

- **.aspx:** A web form that may include a code-behind file (.vb/.cs depending on the code used).
- **.asax:** This file allows you to write code to handle global ASP.NET application-level events. The file has a name of Global.asax, which you cannot change.
- **.ashx:** A page for implementing a generic handler.
- **.asmx:** An ASP.NET web service; it may include a corresponding code-behind file as well includes its code.
- **.htm:** A standard HTML page.
- **.css:** A Cascading Style Sheet that may be used within the site.
- **.sitemap:** A Web application's site map.
- **.skin:** A file used to define an ASP.NET theme that may be used in the site.
- **.browser:** A browser definition file.

- **.disco:** An optional file that acts as a discovery mechanism for the XML Web service. The .disco file is not automatically created for an XML Web service.

- **.ascx:** A Web user control.

You may also use other files that are not on this list depending on whether an application is compiled or how it is deployed.

Imports Statement (.NET Namespace and Type)

Namespace is the way of organizing .Net class library into a logical grouping based on their functionality, usability or category. The namespace is a way of grouping logical types for the purpose of identification and reducing the chance of name collisions. There is a collection of classes in the .Net Framework Class Library (FCL) and have been organized in a hierarchical tree. The root namespace for all the built-in functionality comes under system namespace and all other namespaces come under this. Any class in the .Net Framework Class Library (FCL) can be uniquely identified using the full namespace of the class. For example, System.IO is a logical group of all input and output related features.

Importing namespace enables users to use the elements or methods from that namespace without qualifying the elements in the code. For example, *Create* is a method of the class *System.Messaging.MessageQueue*. By importing the *System.Messaging* namespace user can access the Create method as *MessageQueue.Create*

The namespace can be declared using the keyword *import* in VB and using in C# in codes as follows:

- Namespace Declaration in C#
 - o using System;
 - o using System.Data
- Namespace Declaration in VB
 - o imports system;
 - o imports system.Data

Namespace can be assigned alias name to further reduce the complexity as shown below:

Syntax:

Imports [aliasname =] namespace

-or-

Imports [aliasname =] namespace.element

The following table defines the various terms used in the syntax.

NOTES

NOTES

Term	Definition
aliasname	Optional. An <i>import alias</i> or name by which code can refer to namespace instead of the full qualification string.
namespace	Required. The fully qualified name of the namespace being imported. Can be a string of namespaces nested to any level.
element	Optional. The name of a programming element declared in the namespace. Can be any container element.

The `Imports` statement enables elements or methods to be referenced directly that are contained in a given namespace. User can apply a single namespace name or a string of nested namespaces. Each nested namespace is separated from the next higher level namespace by a period (.), as the following example illustrates.

```
Imports System.Collections.Generic
```

Each source document can contain any number of `Imports` statements. These statements must contain an option declaration such as the `Option Strict` statement preceded by any programming element declaration such as `Module` or `Class` statement.

Import can be used only at file level. The following codes will differentiate between the uses of *import* statement. The first part is written without using import statement as top lines whereas the second set of code uses *import* statement as top lines. Note the difference in referencing `DirectoryInfo`, `StringBuilder`, `CrLf` that have been qualified with the full namespace.

Code without using *import* statement:

```
Public Function GetFolders() As String
    'Create a new StringBuilder, which is used
    'to efficiently build strings.
    Dim sb As New System.Text.StringBuilder

    Dim dInfo As New System.IO.DirectoryInfo("c:\")
    'Obtain an array of directories, and iterate through
    'the array.
    For Each dir As System.IO.DirectoryInfo In
        dInfo.GetDirectories()
        sb.Append(dir.Name)
        sb.Append(Microsoft.VisualBasic.ControlChars.CrLf)
    Next

    Return sb.ToString
End Function
```

Code using *import* statement:

```
'Place Imports statements at the top of your program.
Imports System.Text
Imports System.IO
```

```
Imports Microsoft.VisualBasic.ControlChars
```

Introduction to ASP.NET

```
Public Function GetFolders() As String
```

```
    Dim sb As New StringBuilder
```

```
    Dim dInfo As New DirectoryInfo("c:\")
```

```
        For Each dir As DirectoryInfo In
```

```
dInfo.GetDirectories()
```

```
            sb.Append(dir.Name)
```

```
            sb.Append(CrLf)
```

```
    Next
```

```
    Return sb.ToString
```

```
End Function
```

Global.asax

Global.asax is an optional file which is used to handle higher level application events such as Application_Start, Application_End, Session_Start, Session_End etc raised by ASP.NET or by HttpModules. It is also popularly known as ASP.NET Application File. This file resides in the root directory of an ASP.NET based application and is parsed and compiled by ASP.NET

Global.asax contains a class representing your application as a whole. During run time, this file is parsed and compiled into a dynamically generated .NET framework class derived from the `HttpApplication` base class. This file can be deployed as an assembly in the bin directory of an ASP.NET application. The Global.asax file itself is configured so that if a user requests the file, the request is rejected. External users cannot download or view the code written within it. This file need not be recompiled if there is no changes in the file. There can only be one file per application which should be located in the application's root directory only as stated above.

The Global.asax can handle two types of events:

1. Events which are fired for every request
2. Events which are not fired for every request

Following are the events which are fired for every request.

- **Application_BeginRequest()** – This event raised at the start of every request for the web application.
- **Application_AuthenticateRequest** – This event rose just before the user credentials are authenticated. We can specify our own authentication logic here to provide custom authentication.
- **Application_AuthorizeRequest()** – This event raised after successful completion of authentication with user's credentials. This event

NOTES

NOTES

is used to determine user permissions. You can use this method to give authorization rights to user.

- **Application_ResolveRequestCache()** – This event raised after completion of an authorization request and this event used in conjunction with output caching. With output caching, the rendered HTML of a page is reused without executing its code.
- **Application_AcquireRequestState()** – This event raised just before session-specific data is retrieved for the client and is used to populate Session Collection for current request.
- **Application_PreRequestHandlerExecute()** – This event is called before the appropriate HTTP handler executes the request.
- **Application_PostRequestHandlerExecute()** – This event called just after the request is handled by its appropriate HTTP handler.
- **Application_ReleaseRequestState()** – This event raised when session specific information is about to serialized from the session collection.
- **Application_UpdateRequestCache()** – This event raised just before information is added to output cache of the page.
- **Application_EndRequest()** – This event raised at the end of each request right before the objects released.

Following are the events which are not fired for every request.

- **Application_Start()** – This event is raised when the application starts up and application domain is created.
- **Session_Start()** – This event is raised for each time a new session begins, This is a good place to put code that is session-specific.
- **Application_Error()** – This event raised whenever an unhandled exception occurs in the application. This provides an opportunity to implement generic application-wide error handling.
- **Session_End()** – This event called when session of user ends.
- **Application_End()** – This event raised just before when web application ends.
- **Application_Disposed()** – This event fired after the web application is destroyed. This event is used to reclaim the memory it occupies.

`Global.asax` file does not create normally. You need to add it by yourself. Following are the steps to create `Global.asax` file.

1. Open visual studio
2. Create a new website
3. Go to the solution explorer
4. Add new item

5. Global application class

6. Add the following code in Global.asax file

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
    }
    void Application_End(object sender, EventArgs e)
    {
        // Code that runs on application shutdown
    }
    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs
    }
    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
    }
    void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends.
        // Note: The Session_End event is raised only
        // when the sessionstate mode
        // is set to InProc in the Web.config file. If
        // session mode is set to StateServer
        // or SQLServer, the event is not raised.
    }
</script>
```

NOTES

After that you need to add a class in your project. Following are the steps:

1. Right click App_code
2. Add new item
3. Select class
4. Name it as Global.cs
5. Add the following code

Inherit the newly generated by System.Web.HttpApplication and copy all the method created Global.asax to Global.cs and also add an inherit attribute to the Global.asax file. Now your Global.asax will be as follows:

NOTES

```

public class Global : System.Web.HttpApplication
{
    public Global()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
    }

    /// Many other events like begin request...e.t.c, e.t.c
}

```

Check Your Progress

1. What is namespace?
2. Write the syntax for importing the namespace.
3. Write a note on Global.asax.

10.3 PAGE CLASS

Page class represents an .aspx file, also known as a Web Forms page, requested from a server that hosts an ASP.NET Web application. The page class provides useful properties and methods that can be used in the code. In fact, all the web forms are the instances of the ASP.NET page class and is defined in the namespace System.Web.UI. The Page class is inherited from the TemplateControl class which in turn inherited from the Control class.

Various objects associated with the Page class are given below:

- 1. Session:** An instance of the System.Web.SessionState.HttpSessionState class, it acts as a repository to store any type of user specific data that needs to stay between web-page requests. This object stores data in the form of name/value pairs that maintains the user data such as user's name, ID and other elements which is discarded as soon as the user logs out or is not accessing that web page.
- 2. Application:** This object is also an instance of the System.Web.HttpApplicationState class and stores data in the form of name/value pairs. But the scope of this object is for the entire application.
- 3. Cache:** is an instance of the System.Web.Caching.Cache class which also stores global information in the form of name/value pair but the expiration policy and dependencies can be customized for each item so that items can be removed automatically when the resources (such as files, database tables etc.) are modified or updated.

4. **Request:** This is an instance of the System.Web.HttpRequest class that contains all the values and properties of the HTTP request that causes the page to be loaded. It also contains other information such as URL parameter sent by the client.

HttpRequest Properties

- **ApplicationPath and PhysicalPath:** ApplicationPath gets the ASP.NET application's virtual application root path on server while PhysicalPath gets the Physical file system path corresponding to the requested URL.
- **AnonymousID:** it uniquely identifies the current user if we have enabled anonymous access.
- Browser provides a link to the requesting client's browser capabilities object.
- ClientCertificate is an HttpClientCertificate object that gets the security certificate for the current request.
- Cookies gets the collection cookies sent with this request.
- FilePath and CurrentExecutionFilePath return the virtual path of the current request.
- Form represents the collection of form variables that were posted back to the page. In almost all cases we will retrieve the information from control properties instead of using this collection
- ServerVariables returns a collection of named server variables sent with the request.
- IsAuthenticated returns true if the user has been successfully authenticated.
- IsSecureConnection indicates whether the HTTP connection uses secure sockets (that is, https).
- IsLocal returns true if the user is requesting the page from the current system.
- QueryString provides the parameters that are passed along with the query string.
- UrlReferrer provides a URL object that represents the current address for the page where the user is coming from (the previous page that linked to this page).
- UserAgent is a string representing the browser type.
- UserHostAddress and UserHostName get the IP address and the DNS name of the remote client.

NOTES

NOTES

- UserLanguages provides a stored string array of client's language preference. This can be useful if we need to create multilingual pages.

5. Response: It is an object and an instance of the System. Web.HttpResponse class that represents the web server's response to a client request.

HttpResponse Properties:

- Buffer Output, when set to true (the default), the page is not sent to the client until the entire page is finished processing.
- Cache allows us to configure output caching of a Web page.
- Cookies is the collection of cookies sent with the response.
- Expires and Expires Absolute properties are used to cache the rendered HTML for the page for a specified period of time, which helps to improve performance for subsequent requests.
- IsClientConnected is a Boolean value which indicates whether the client is still connected to the server.
- Redirect () method is used to transfers the user to another page in the application or to a different web site.
- ContentType is a header that tells the browser what type of content it is about to receive. In general, ASP.NET web forms use text or html content type. However, we can create a custom HTTP handler that serves different types of content.
- OutputStream represents the data we are sending to the browser as a stream of raw bytes.
- Write () method allows us to write text directly to the response stream.
- BinaryWrite () and WriteFile () methods allow to take binary content from a byte array or form a file and write it directly to the response stream.

6. Server object is an instance of the System.Web.HttpServerUtility class. It provides various methods and properties as discussed below:

- MachineName is a property representing the computer name of the computer on which the page is running, that means the server.
- CreateObject() creates an instance of the COM object that is identified by the object's programmatic identifier (ProgID).
- GetLastError retrieves the exception object for the most recently encountered error. This is generally used in an application event handler that checks for error conditions.

C#

```
Public class Page : System.Web.UI.TemplateControl,
System.Web.IHttpHandler
```

The accompanying code precedent exhibits how the Page class is utilized in the code-behind page demonstrate. Note that the code-behind source document announces a fractional class that acquires from a base page class. The base page class can be Page, or it very well may be another class that gets from Page. Moreover, note that the halfway class enables the code-behind record to utilize controls characterized on the page without the need to characterize them as field individuals. The following code example shows the .aspx file that corresponds to the preceding code-behind source file.

ASP.NET (C#)

```
<%@ Page Language="C#" CodeFile="pageexample.aspx.cs"
Inherits="MyCodeBehindCS" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Page Class Example</title>
</head>
<body>
    <form id="form1" 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>
```

NOTES

NOTES

You must use the directive and use the Inherits and CodeFile attributes to link the code-behind file to the .aspx file. In this example, the Inherits attribute indicates the MyCodeBehind class and the CodeFile attribute indicates the path to the language-specific file that contains the class.

10.4 HttpRequest

As discussed above, `HttpRequest` is an instance of the `System.Web.HttpRequest` class that contains all the values and properties of the HTTP request that causes the page to be loaded. It also contains other information such as URL parameter, client browser, cookies etc. sent by the client. It enables ASP.NET to read the HTTP values sent by a client during a Web request. Some noteworthy properties of this object have been discussed above.

Following are the important methods of `HttpRequest` class:

1. `BinaryRead` performs a binary read of a specified number of bytes from the current input stream.
2. `Equals(Object)` determines whether the specified object is equal to the current object.
3. `GetType` gets the type of the current instance.
4. `MapImageCoordinates` maps an incoming image-field form parameter to appropriate x-coordinate and y-coordinate values.
5. `MapPath(String)` maps the specified virtual path to a physical path.
6. `SaveAs` saves an HTTP request to disk.
7. `ToString` returns a String that represents the current object.
8. `ValidateInput` causes validation to occur for the collections accessed through the Cookies, Form, and QueryString properties.

The following examples access the `HttpRequest` instance for the current request by using the `Request` property of the `Page` class. You can use simplified syntax for accessing data from the `QueryString`, `Form`, `Cookies`, or `ServerVariables` collections.

The code below shows how to retrieve a query string value when loading a page.

C#

```
public sealed class HttpRequest
public partial class AddToCart : Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
```

```

        string rawId = Request["ProductID"];
        int productId;
        if (!String.IsNullOrEmpty(rawId) &&
int.TryParse(rawId, out productId))
    {
        using (ShoppingCartActions
usersShoppingCart = new ShoppingCartActions())
    {
        usersShoppingCart.AddToCart(productId);
    }
}
else
{
    throw new Exception("Tried to call AddToCart.aspx
without setting a ProductId.");
}
Response.Redirect("ShoppingCart.aspx");
}
}

```

The code below shows how to check if the request is authenticated and retrieve the raw URL.

C#

```

public partial class RestrictedPage : Page
{
    protected void Page_Load(object sender, EventArgs
e)
    {
        if (!Request.IsAuthenticated)
        {
            var rawUrl = Request.RawUrl;
            Response.Redirect("/Account/Login?ru=" +
Server.HtmlEncode(rawUrl));
        }
    }
}

```

This example code uses the StreamWriter class to write the values of several HttpRequest class properties to a file. For properties that are of type string, the values are HTML encoded as they are written to the file. Properties that represent a collection are looped through, and each key/value pair that they contain is written to the file.

NOTES

NOTES

10.5 **HttpResponse CLASS**

As discussed earlier, Response is an object and an instance of the System.Web.HttpResponse class that represents the web server's response to a client request. In ASP.NET, the response object of this class does not play a vital role in sending HTML text to the client as the server-side controls have nested, object oriented methods for rendering themselves. But the object still provides some important properties such as the Redirect() method that allows the user to transfer to other page both inside and outside the application. Following are the important methods associated with this class:

1. **AddHeader:** Adds an HTTP header to the output stream. AddHeader is provided for compatibility with earlier versions of ASP.
2. **AppendCookie:** Infrastructure adds an HTTP cookie to the intrinsic cookie collection.
3. **AppendHeader:** Adds an HTTP header to the output stream.
4. **AppendToLog:** Adds custom log information to the InterNET Information Services (IIS) log file.
5. **BinaryWrite:** Writes a string of binary characters to the HTTP output stream.
6. **ClearContent:** Clears all content output from the buffer stream.
7. **Close:** Closes the socket connection to a client.
8. **End:** Sends all currently buffered output to the client, stops execution of the page, and raises the EndRequest event.
9. **Equals(Object):** Determines whether the specified object is equal to the current object.
10. **Flush:** Sends all currently buffered output to the client.
11. **GetType:** Gets the Type of the current instance.
12. **Pics:** Appends a HTTP PICS-Label header to the output stream.
13. **Redirect(String):** Redirects a request to a new URL and specifies the new URL.
14. **Redirect(String, Boolean):** Redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.
15. **SetCookie:** Updates an existing cookie in the cookie collection.
16. **ToString:** Returns a String that represents the current Object.
17. **TransmitFile(String):** Writes the specified file directly to an HTTP response output stream, without buffering it in memory.
18. **Write(Char):** Writes a character to an HTTP response output stream.

19. **Write(Object)**: Writes an object to an HTTP response stream.
 20. **Write(String)**: Writes a string to an HTTP response output stream.
 21. **WriteFile(String)**: Writes the contents of the specified file directly to an HTTP response output stream as a file block.
 22. **WriteFile(String, Boolean)**: Writes the contents of the specified file directly to an HTTP response output stream as a memory block.
- Encapsulates HTTP-response information from an ASP.NET operation.

NOTES**C#**

```
public sealed class HttpResponse
```

The following example draws three overlapping rectangles when the page is requested. In this code, we will set the the ContentType property to picture/jpeg, with the goal that the whole page will be rendered as a JPEG picture. The code at that point calls the Clear strategy to guarantee that no incidental substance is sent with this reaction. Next, the code sets the BufferOutput property to genuine with the goal that the page is totally handled before it is sent to the mentioning customer. Two items used to draw the rectangular shapes are then made a Bitmap and a Graphics object. The factors made in the page are utilized as directions to draw the rectangular shapes and a string that shows up inside the larger rectangular shape.

The Bitmap is saved to the Stream object that is related with the OutputStream property and its arrangement is set to JPEG, when the three rectangular shapes and the string that shows up inside them are drawn. The code calls the Dispose and Dispose methods to release the assets acquired by the two objects. The code calls the Flush methods to send the buffered response to the requesting client.

ASP.NET (C#)

```
<%@ Page Language="C#" %>
<%@ import Namespace="System.Drawing" %>
<%@ import Namespace="System.Drawing.Imaging" %>
<%@ import Namespace="System.Drawing.Drawing2D" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<script runat="server">

    private void Page_Load(object sender, EventArgs e)
    {
        // <snippet2>
        // Set the page's content type to JPEG files
```

NOTES

```
// and clears all content output from the buffer
stream.

Response.ContentType = "image/jpeg";
Response.Clear();

// Buffer response so that page is sent
// after processing is complete.
Response.BufferOutput = true;
// </snippet2>

// Create a font style.
Font rectangleFont = new Font(
    "Arial", 10, FontStyle.Bold);

// Create integer variables.
int height = 100;
int width = 200;

// Create a random number generator and create
// variable values based on it.
Random r = new Random();
int x = r.Next(75);
int a = r.Next(155);
int x1 = r.Next(100);

// Create a bitmap and use it to create a
// Graphics object.
Bitmap bmp = new Bitmap(
    width, height, PixelFormat.Format24bppRgb);
Graphics g = Graphics.FromImage(bmp);

g.SmoothingMode = SmoothingMode.AntiAlias;
g.Clear(Color.LightGray);

// Use the Graphics object to draw three
rectangles.
g.DrawRectangle(Pens.White, 1, 1, width-3,
height-3);
g.DrawRectangle(Pens.Aquamarine, 2, 2, width-
```

```

    3, height-3);
            g.DrawRectangle(Pens.Black, 0, 0, width,
height);

        // Use the Graphics object to write a string
        // on the rectangles.
        g.DrawString(
            "ASP.NET Samples", rectangleFont,
SystemBrushes.WindowText, new PointF(10, 40));

        // Apply color to two of the rectangles.
        g.FillRectangle(
            new SolidBrush(
                Color.FromArgb(a, 255, 128, 255)),
x, 20, 100, 50);

        g.FillRectangle(
            new LinearGradientBrush(
                new Point(x, 10),
                new Point(x1 + 75, 50 + 30),
                Color.FromArgb(128, 0, 0, 128),
                Color.FromArgb(255, 255, 255, 240)),
x1, 50, 75, 30);

// <snippet3>
        // Save the bitmap to the response stream and
        // convert it to JPEG format.
        bmp.Save(Response.OutputStream,
ImageFormat.Jpeg);

        // Release memory used by the Graphics object
        // and the bitmap.
        g.Dispose();
        bmp.Dispose();

        // Send the output to the client.
        Response.Flush();
// </snippet3>

```

Introduction to ASP.NET

NOTES

```
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>ASP.NET Example</title>
</head>
<body>
    <form id="form1" runat="server">
    </form>
</body>
</html>
```

NOTES

10.6 SERVER UTILITY

HttpServerUtility class provides methods for processing web requests. It performs utility functions such as encoding and decoding strings for use in URLs or for plain-text display of content that may contain HTML mark-up tags. It also provides access to some error information, provides methods to modify the execution of the current request. The server object is an instance of System.Web.HttpServerUtility class.

Following are some important properties associated with this class:

- **MachineName:** Name of server computer.
- **ScriptTimeOut:** Gets and sets the request time-out value in seconds.

Following are some methods available in System.Web.HttpServerUtility class:

- **CreateObject(String):** Creates an instance of the COM object identified by its ProgID (Programmatic ID).
- **CreateObject(Type):** Creates an instance of the COM object identified by its Type.
- **Equals(Object):** Determines whether the specified Object is equal to the current Object.
- **Execute(String):** Executes the handler for the specified virtual path in the context of the current request.
- **Execute(String, Boolean):** Executes the handler for the specified virtual path in the context of the current request and specifies whether to clear the QueryString and Form collections.
- **GetLastError:** Returns the previous exception.
- **GetType:** Gets the Type of the current instance.
- **HtmlEncode:** Changes an ordinary string into a string with legal HTML characters.

- **HtmlDecode:** Converts an Html string into an ordinary string.
- **ToString:** Returns a String that represents the current Object.
- **Transfer(String):** For the current request, terminates execution of the current page and starts execution of a new page by using the specified URL path of the page.
- **UrlDecode:** Converts an URL string into an ordinary string.
- **UrlEncodeToken:** Works same as UrlEncode, but on a byte array that contains Base64-encoded data.
- **UrlDecodeToken:** Works same as UrlDecode, but on a byte array that contains Base64-encoded data.
- **MapPath:** Return the physical path that corresponds to a specified virtual file path on the server.
- **Transfer:** Transfers execution to another web page in the current application.

NOTES

The following code (written in c#) demonstrates how to use the HtmlEncode method and the UrlEncode method of the HttpServerUtility class. The HtmlEncode method helps ensure that any user-supplied string input will be rendered as static text in browsers instead of executable script or HTML elements. The UrlEncode method encodes URLs, so that they are correctly transmitted in the HTTP stream.

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">

<script runat="server">

    protected void Button1_Click(object sender,
EventArgs e)
    {
        if (!String.IsNullOrEmpty(TextBox1.Text))
        {
            // Access the HttpServerUtility methods
            // through
            // the intrinsic Server object.
            Label1.Text = "Welcome, " +
                Server.HtmlEncode(TextBox1.Text) +
                ".<br/> The url is " +
                Server.UrlEncode(Request.Url.ToString());
        }
    }
}
```

```
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>HttpServerUtility Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Enter your name:<br />

            <asp:TextBox ID="TextBox1" runat="server"></
asp:TextBox>
            <asp:Button ID="Button1" runat="server"
OnClick="Button1_Click" Text="Submit" />
            <br />
            <asp:Label ID="Label1" runat="server"/>
        </div>
    </form>
</body>
</html>
```

NOTES**Check Your Progress**

4. Write a note on HttpRequest.
5. What does HTTPServerUtility class provides?

10.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Namespace is the way of organizing .Net class library into a logical grouping based on their functionality, usability or category. The namespace is a way of grouping logical types for the purpose of identification and reducing the chance of name collisions.
2. The syntax for importing the namespace is as follows:

Imports [aliasname =] namespace
-or-
Imports [aliasname =] namespace.element
3. Global.asax is an optional file which is used for handling higher level application events such as Application_Start, Application_End,

Session_Start, Session_End etc. It is also popularly known as ASP.NET Application File.

Introduction to ASP.NET

4. `HttpRequest` is an instance of the `System.Web.HttpRequest` class that contains all the values and properties of the HTTP request that causes the page to be loaded. It also contains other information such as URL parameter, client browser, cookies etc. sent by the client.
5. `HttpServerUtility` class provides methods for processing web requests. It performs utility functions such as encoding and decoding strings for use in URLs or for plain-text display of content that may contain HTML mark-up tags.

NOTES

10.8 SUMMARY

- Active Server Pages (ASP) is a file extension for the file format used by an HTML file that contains a script processed by Microsoft server and is designed for ASP.Net framework.
- There are variety of languages available to write the ASP.Net applications code such as C#, VB.Net, Jscript, J#. Any of the language can be used to develop web applications.
- CLR is responsible for managing various key activities such as memory management, exception handling, security checking, garbage collection, debugging, thread execution, code execution, code safety, verification and compilation.
- ASP.NET development uses specific file types depending on the resource used.
- Namespace is the way of organizing .Net class library into a logical grouping based on their functionality, usability or category. The namespace is a way of grouping logical types for the purpose of identification and reducing the chance of name collisions.
- The `Imports` statement enables elements or methods to be referenced directly that are contained in a given namespace. User can apply a single namespace name or a string of nested namespaces.
- `Global.asax` is an optional file which is used to handle higher level application events such as `Application_Start`, `Application_End`, `Session_Start`, `Session_End` etc raised by ASP.NET or by HttpModules.
- Page class represents an `.aspx` file, also known as a Web Forms page, requested from a server that hosts an ASP.NET Web application.
- `HttpRequest` is an instance of the `System.Web.HttpRequest` class that contains all the values and properties of the HTTP request that causes

NOTES

the page to be loaded. It also contains other information such as URL parameter, client browser, cookies etc. sent by the client.

- Response is an object and an instance of the System.Web.HttpResponse class that represents the web server's response to a client request.
- HttpServerUtility class provides methods for processing web requests. It performs utility functions such as encoding and decoding strings for use in URLs or for plain-text display of content that may contain HTML mark-up tags. It also provides access to some error information, provides methods to modify the execution of the current request.

10.9 KEY WORDS

- **Global.asax:** It is an optional file which is used for handling higher level application events such as Application_Start, Application_End, Session_Start, Session_End etc.
- **HttpRequest:** It enables ASP.NET to read the HTTP values sent by a client during a Web request.
- **HttpResponse:** It contains methods and properties necessary for crafting an HTTP response.

10.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Define the term namespace.
2. How will you import the namespace in ASP.NET?
3. Discuss the significance of Global.Asax file.
4. What do you understand by SQL server utility?
5. Discuss some of methods available in System.Web.HttpServer Utility class.

Long Answer Questions

1. What are the various component on which the basic architecture of the ASP.Net framework is based?
2. What are the various types of files in ASP.NET?
3. Explain the various objects associated with the Page class.

4. Write a code to illustrate the use of `HttpRequest` class.
5. Write a code to illustrate the use of `HttpResponse` class.
6. What are the important methods associated with `HttpResponse` class?

Introduction to ASP.NET

NOTES

10.11 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 11 BASIC WEB CONTROLS

NOTES

Structure

- 11.0 Introduction
 - 11.1 Objectives
 - 11.2 Web Controls
 - 11.2.1 Properties of the Server Controls
 - 11.2.2 Methods of the Server Controls
 - 11.3 ListControl Class
 - 11.4 Validation Controls
 - 11.5 Rich Controls
 - 11.6 Data Bound Controls
 - 11.7 Custom Controls
 - 11.8 Answers to Check Your Progress Questions
 - 11.9 Summary
 - 11.10 Key Words
 - 11.11 Self Assessment Questions and Exercises
 - 11.12 Further Readings
-

11.0 INTRODUCTION

In this unit, you will learn about the web form controls. ASP.Net provides web forms controls which are categorized as server and client based. Controls are the basic building blocks of the Graphical User Interface (GUI) that enables users to interact with the user by allowing them to enter data, make selections to indicate their preference and to display data. We generally work with Web server controls. The most basic and frequently used web server controls are text boxes, buttons, check boxes, list boxes, and radio buttons.

11.1 OBJECTIVES

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

- Discuss the various categories of web controls
 - Explain the properties and methods associated with server controls
 - Explain and work with the different list, validation, rich, data and custom controls
-

11.2 WEB CONTROLS

ASP.NET uses five types of web controls, which are as follows:

1. HTML controls
2. HTML Server controls

3. ASP.NET Server controls
4. ASP.NET Ajax Server controls
5. User controls and custom controls

Basic Web Controls

ASP.NET server controls are the primary controls used in ASP.NET and are also known as web controls. These controls can be grouped into the following categories:

1. **Validation controls:** These are used to validate user input and they work by running client-side script.
2. **Data source controls:** These controls provides data binding to different data sources.
3. **Data view controls:** These are various lists and tables, which can bind to data from data sources for displaying.
4. **Personalization controls:** These are used for personalization of a page according to the user preferences, based on user information.
5. **Login and security controls:** These controls provide user authentication.
6. **Master pages:** These controls provide consistent layout and interface throughout the application.
7. **Navigation controls:** These controls help in navigation. For example, menus, tree view etc.
8. **Rich controls:** These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

A special tag is used to define ASP.Net web controls which starts with “*<asp:*” prefix followed by the corresponding control class name. for example, the tag used for TextBox controls is *<asp:TextBox>* and the tag used for Label controls is “*<asp:Label>*”. There are other attributes (eg. ID, Text (in case of label) etc.) also that are present in the tag which are actually the properties of the selected control. These properties (attributes) are also listed in the properties window. **ID**, which has to be unique per page and **runat=“server”**, that indicates that the control will be processed by the ASP.Net web server are the mandatory attributes for every ASP.Net control.

The following syntax is used for the server controls:

```
<asp:controlType ID = "ControlID" runat="server"  
Property1=value1 [Property2=value2] />
```

In addition, visual studio has the following features, to help produce an error-free coding:

1. Dragging and dropping of controls in design view
2. IntelliSense feature that displays and auto-completes the properties
3. The properties window to set the property values directly

NOTES

NOTES**11.2.1 Properties of the Server Controls**

ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class. The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

The following table shows the inherited properties, common to all server controls.

Properties	Description
AccessKey	Pressing this key with the Alt key moves focus to the control.
Attributes	It is the collection of arbitrary attributes for rendering only that do not correspond to properties on the control.
BackColor	Background color.
BindingContainer	The control that contains this control's data binding.
BorderColor	Border color.
BorderStyle	Border style.
BorderWidth	Border width.
CausesValidation	Indicates if it causes validation.
ChildControlCreated	It indicates whether the server control's child controls have been created.
ClientID	Control ID for HTML markup.
Context	The HttpContext object associated with the server control.
Controls	Collection of all controls contained within the control.
ControlStyle	The style of the Web server control.
CssClass	CSS class
DataItemContainer	Gets a reference to the naming container if the naming container implements IDataItemContainer.
DataKeysContainer	Gets a reference to the naming container if the naming container implements IDataKeysControl.
DesignMode	It indicates whether the control is being used on a design surface.
DisabledCssClass	Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.
Enabled	Indicates whether the control is grayed out.
EnableTheming	Indicates whether theming applies to the control.
EnableViewState	Indicates whether the view state of the control is maintained.
Events	Gets a list of event handler delegates for the control.
Font	Font.
Forecolor	Foreground color.
HasAttributes	Indicates whether the control has attributes set.
HasChildViewState	Indicates whether the current server control's child controls have any saved view-state settings.
Height	Height in pixels or %.

ID	Identifier for the control.
IsChildControlStateCleared	Indicates whether controls contained within this control have control state.
IsEnabled	Gets a value indicating whether the control is enabled.
IsTrackingViewState	It indicates whether the server control is saving changes to its view state.
IsViewStateEnabled	It indicates whether view state is enabled for this control.
LoadViewStateById	It indicates whether the control participates in loading its view state by ID instead of index.
Page	Page containing the control.
Parent	Parent control.
RenderingCompatibility	It specifies the ASP.NET version that the rendered HTML will be compatible with.
Site	The container that hosts the current control when rendered on a design surface.
SkinID	Gets or sets the skin to apply to the control.
Style	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
TabIndex	Gets or sets the tab index of the Web server control.
TagKey	Gets the HtmlTextWriterTag value that corresponds to this Web server control.
TagName	Gets the name of the control tag.
TemplateControl	The template that contains this control.
TemplateSourceDirectory	Gets the virtual directory of the page or control containing this control.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the web server control.
UniqueID	Unique identifier.
ViewState	Gets a dictionary of state information that saves and restores the view state of a server control across multiple requests for the same page.
ViewStateIgnoreCase	It indicates whether the StateBag object is case-insensitive.
ViewStateMode	Gets or sets the view-state mode of this control.
Visible	It indicates whether a server control is visible.
Width	Gets or sets the width of the Web server control.

NOTES**11.2.2 Methods of the Server Controls**

The following table provides the methods of the server controls.

Methods	Description
AddAttributesToRender	Adds HTML attributes and styles that need to be rendered to the specified HtmlTextWriterTag.
AddedControl	Called after a child control is added to the Controls collection of the control object.

<i>Basic Web Controls</i>	AddParsedSubObject	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's control collection.
NOTES	ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.
	ClearCachedClientID	Infrastructure. Sets the cached ClientID value to null.
	ClearChildControlState	Deletes the control-state information for the server control's child controls.
	ClearChildState	Deletes the view-state and control-state information for all the server control's child controls.
	ClearChildViewState	Deletes the view-state information for all the server control's child controls.
	CreateChildControls	Used in creating child controls.
	CreateControlCollection	Creates a new ControlCollection object to hold the child controls.
	CreateControlStyle	Creates the style object that is used to implement all style related properties.
	DataBind	Binds a data source to the server control and all its child controls.
	DataBindBooleanBoolean	Binds a data source to the server control and all its child controls with an option to raise the DataBinding event.
	DataBindChildren	Binds a data source to the server control's child controls.
	Dispose	Enables a server control to perform final clean up before it is released from memory.
	EnsureChildControls	Determines whether the server control contains child controls. If it does not, it creates child controls.
	EnsureID	Creates an identifier for controls that do not have an identifier.
	EqualsObjectObject	Determines whether the specified object is equal to the current object.
	Finalize	Allows an object to attempt to free resources and perform other cleanup operations before the object is reclaimed by garbage collection.
	FindControlStringString	Searches the current naming container for a server control with the specified id parameter.
	FindControlString,Int32String, Int32	Searches the current naming container for a server control with the specified id and an integer.
	Focus	Sets input focus to a control.
	GetDesignModeState	Gets design-time data for a control.
	GetType	Gets the type of the current instance.
	GetUniqueIDRelativeTo	Returns the prefixed portion of the UniqueID property of the specified control.
	HasControls	Determines if the server control contains any child controls.
	HasEvents	Indicates whether events are registered for the control or any child controls.
	IsLiteralContent	Determines if the server control holds only literal content.

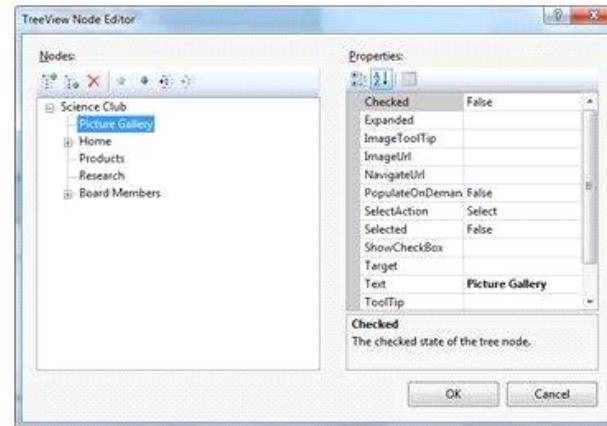
LoadControlState	Restores control-state information.	<i>Basic Web Controls</i>
LoadViewState	Restores view-state information.	
MapPathSecure	Retrieves the physical path that a virtual path, either absolute or relative, maps to.	
MemberwiseClone	Creates a shallow copy of the current object.	NOTES
MergeStyle	Copies any nonblank elements of the specified style to the web control, but does not overwrite any existing style elements of the control.	
OnBubbleEvent	Determines whether the event for the server control is passed up the page's UI server control hierarchy.	
OnDataBinding	Raises the data binding event.	
OnInit	Raises the Init event.	
OnLoad	Raises the Load event.	
OnPreRender	Raises the PreRender event.	
OnUnload	Raises the Unload event.	
OpenFile	Gets a Stream used to read a file.	
RemovedControl	Called after a child control is removed from the controls collection of the control object.	
Render	Renders the control to the specified HTML writer.	
RenderBeginTag	Renders the HTML opening tag of the control to the specified writer.	
RenderChildren	Outputs the contents of a server control's children to a provided HtmlTextWriter object, which writes the contents to be rendered on the client.	
RenderContents	Renders the contents of the control to the specified writer.	
RenderControlHtmlTextWriter	Outputs server control content to a provided HtmlTextWriter object and stores tracing information about the control if tracing is enabled.	
HtmlTextWriter		
RenderEndTag	Renders the HTML closing tag of the control into the specified writer.	
ResolveAdapter	Gets the control adapter responsible for rendering the specified control.	
SaveControlState	Saves any server control state changes that have occurred since the time the page was posted back to the server.	
SaveViewState	Saves any state that was modified after the TrackViewState method was invoked.	
SetDesignModeState	Sets design-time data for a control.	
ToString	Returns a string that represents the current object.	
TrackViewState	Causes the control to track changes to its view state so that they can be stored in the object's view state property.	

Example 11.1: Tree view control

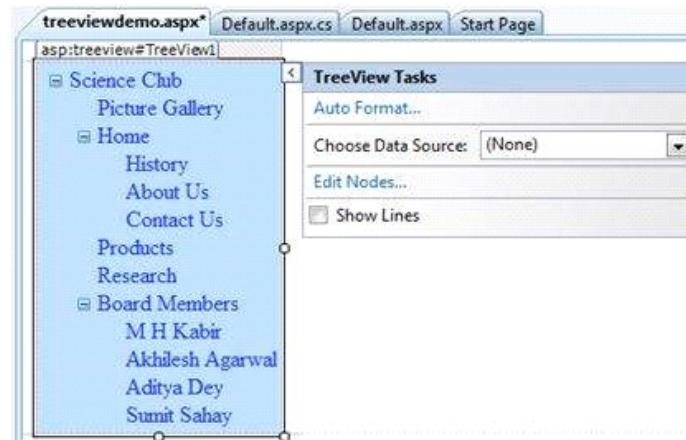
A Tree view control comes under navigation controls. Other Navigation controls are: Menu control and SiteMapPath control.

Add a tree view control on the page. Select Edit Nodes from the tasks. Edit each of the nodes using the Tree view node editor as shown below:

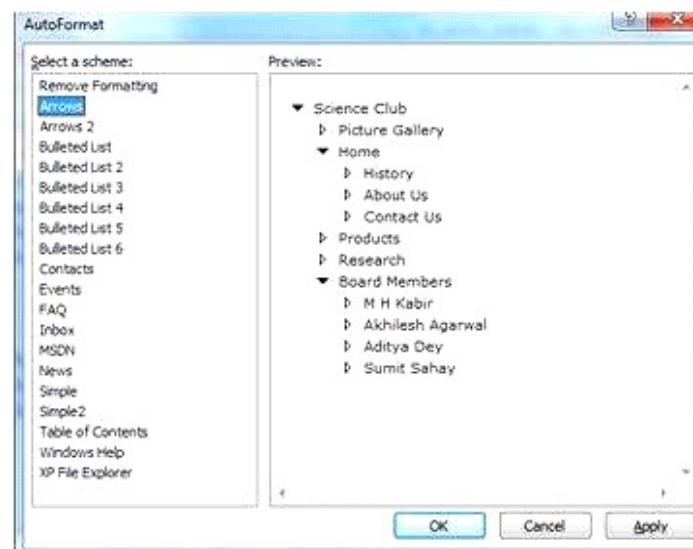
NOTES



Once you have created the nodes, it looks like as follows in design view:



The AutoFormat task allows you to format the tree view as shown below:



Add a label control and a text box control on the page and name them lblmessage and txtmessage respectively.

Write the following code to ensure that when a particular node is selected, the label control displays the node text and the text box displays all child nodes under it, if any.

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {
    public partial class treeviewdemo : System.Web.UI.Page
    {

        protected void Page_Load(object sender, EventArgs e) {
            txtmessage.Text = " ";
        }

        protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e) {
            txtmessage.Text = " ";
            lblmessage.Text = "Selected node changed to: " +
TreeView1.SelectedNode.Text;
            TreeNodeCollection childnodes =
TreeView1.SelectedNode.ChildNodes;

            if(childnodes != null) {
                txtmessage.Text = " ";

```

Basic Web Controls

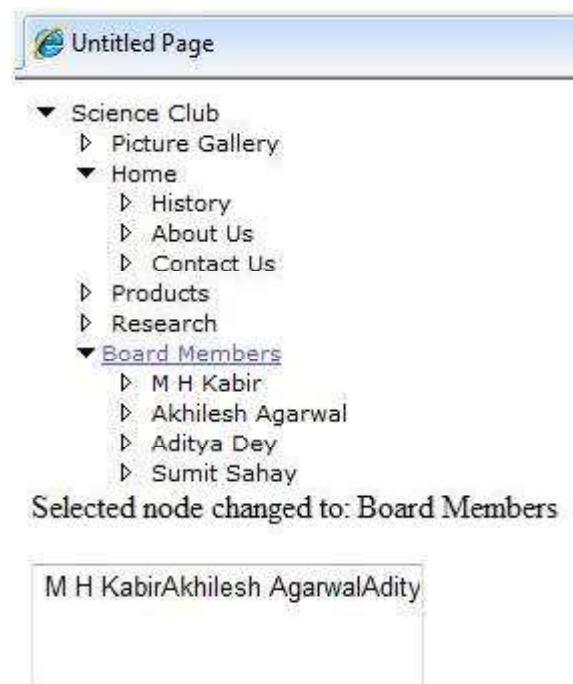
NOTES

```
foreach (TreeNode t in childnodes) {  
    txtmessage.Text += t.Value;  
}
```

NOTES

```
}  
}  
}  
}
```

Execute the page to see the effects. You will be able to expand and collapse the nodes.



11.3 LISTCONTROL CLASS

The collection of items is displayed through the List control that enables to display simple lists of options. ASP.Net provides the following list controls:

1. List Box
2. Drop Down list
3. Radio Button List
4. Check Box List
5. Bulleted List

These controls are used to display list of options to select. The selection can be single or multiple depending on the control. All controls are derived from the class `System.Web.UI.WebControls.ListControl`.

Following are some common properties of list controls:

Basic Web Controls

- SelectedValue: Gets the value of the selected item from the list control.
- SelectedIndex: Gets the index of the selected item from the list control.
- SelectedItem: Gets the text of selected item from the list control.
- Items: Gets the collection of items from the dropdown list control.
- DataTextField: Name of the data source field to supply the text of the items. Generally this field comes from the datasource.
- DataValueField: Name of the data source field to supply the value of the items. This is not visible field to list controls, but can be used in the code.
- DataSourceID: ID of the datasource control to provide data.

NOTES

The following code gives a typical usage of individuals for the ListBox and ComboBox classes. The accompanying code model is also a finished application that indicates how you can utilize DataSource, DisplayMember, ValueMember, and SelectedValue individuals from the ListControl class as executed by the ListBox class.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Collections;

namespace MyListControlSample
{
    public class ListBoxSample3 : Form
    {
        private ListBox ListBox1 = new ListBox();
        private Label label1 = new Label();
        private TextBox textBox1 = new TextBox();

        [STAThread]
        static void Main()
        {
            Application.Run(new ListBoxSample3());
        }

        public ListBoxSample3()
        {
            this.ClientSize = new Size(307, 206);
            this.Text = "ListBox Sample3";
        }
    }
}
```

```
ListBox1.Location = new Point(54, 16);
ListBox1.Name = "ListBox1";
ListBox1.Size = new Size(240, 130);
```

NOTES

```
label1.Location = new Point(14, 150);
label1.Name = "label1";
label1.Size = new Size(40, 24);
label1.Text = "Value";

textBox1.Location = new Point(54, 150);
textBox1.Name = "textBox1";
textBox1.Size = new Size(240, 24);

this.Controls.AddRange(new Control[] {
    ListBox1, label1, textBox1 });

// Populate the list box using an array as
DataSource.
ArrayList USStates = new ArrayList();
USStates.Add(new USState("Alabama", "AL"));
    USStates.Add(new USState("Washington",
"WA"));
    USStates.Add(new USState("West Virginia",
"WV"));
    USStates.Add(new USState("Wisconsin", "WI"));
    USStates.Add(new USState("Wyoming", "WY"));
    ListBox1.DataSource = USStates;

// Set the long name as the property to be
displayed and the short
// name as the value to be returned when a
row is selected. Here
// these are properties; if we were binding
to a database table or
// query these could be column names.
    ListBox1.DisplayMember = "LongName";
    ListBox1.ValueMember = "ShortName";

// Bind the SelectedValueChanged event to
our handler for it.
    ListBox1.SelectedValueChanged +=
        new EventHandler(ListBox1_SelectedValueChanged);
```

```
// Ensure the form opens with no rows selected.  
    ListBox1.ClearSelected();  
}  
  
private void InitializeComponent()  
{  
}  
  
private void ListBox1_SelectedIndexChanged(object  
sender, EventArgs e)  
{  
    if (ListBox1.SelectedIndex != -1)  
    {  
        textBox1.Text =  
ListBox1.SelectedValue.ToString();  
        // If we also wanted to get the displayed  
text we could use  
        // the SelectedItem item property:  
        // string s =  
((USState)ListBox1.SelectedItem).LongName;  
    }  
}  
  
public class USState  
{  
    private string myShortName;  
    private string myLongName;  
  
    public USState(string strLongName, string  
strShortName)  
    {  
  
        this.myShortName = strShortName;  
        this.myLongName = strLongName;  
    }  
  
    public string ShortName  
    {  
        get
```

NOTES

```
{  
    return myShortName;  
}  
}  
}
```

NOTES

```
public string LongName  
{  
  
    get  
    {  
        return myLongName;  
    }  
}  
  
}  
}
```

Check Your Progress

1. Which tag is used to define ASP.Net web controls?
2. Which class is used to derive the list controls?

11.4 VALIDATION CONTROLS

ASP.NET validation controls perform validation on both, the client (or browser) and the server to ensure that useless, unauthenticated, or contradictory data did not get stored.

ASP.NET provides the following validation controls:

1. RequiredFieldValidator
2. RangeValidator
3. CompareValidator
4. RegularExpressionValidator
5. CustomValidator
6. ValidationSummary

Generally, the validation is performed on the client side and validation controls uses JavaScript to perform validation. If browser on the client side does not support JavaScript then server side validation is performed.

Base Validator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. BaseValidator class is an abstract class. Therefore, it would be useful to take a look at the properties and the methods of this base class, which are common for all the validation controls.

NOTES

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
    runat="server" ControlToValidate ="ddlcandidate"
    ErrorMessage="Please choose a candidate"
    InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties which are tabulated below.

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

NOTES

The syntax of the control is:

```
<asp:RangeValidator ID="rvclass" runat="server"
ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)"
MaximumValue="12"
MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control. It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1"
runat="server"
ErrorMessage="CompareValidator">

</asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property. The following table summarizes the commonly used syntax constructs for regular expressions.

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-zA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

NOTES

Quantifiers can be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is:

```
<asp:RegularExpressionValidator ID="string" runat="server"
    ErrorMessage="string"
    ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation. The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

NOTES

The basic syntax for the control is:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
    ClientValidationFunction=".cvf_func."
    ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

Validation Summary

The Validation Summary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- 1. ShowSummary** : It shows the error messages in specified format.
- 2. ShowMessageBox** : It shows the error messages in a separate window.

The syntax for the control is:

```
<asp:ValidationSummary ID="ValidationSummary1"
    runat="server"
    DisplayMode = "BulletList" ShowSummary = "true"
    HeaderText="Errors:" />
```

Validation Groups

Complex pages have diverse gatherings of data given in various boards. In such circumstance, a need may emerge for performing approval independently for discrete gathering. This sort of circumstance is taken care of utilizing approval gatherings. To make an approval gathering, you should put the information controls and the approval controls into the equivalent coherent gathering by setting their ValidationGroup property.

11.5 RICH CONTROLS

ASP.NET provides a large set of controls. These controls are divided into different categories, depending upon their functionalities. Rich controls contain rich functionality and are built with multiple HTML elements. The followings control comes under the rich controls.

1. FileUpload control
2. Calendar control
3. AdRotator control
4. MultiView control
5. Wizard control

FileUpload control is used to browse and upload files. After the file is uploaded, you can store the file on any drive or database. FileUpload control is the combination of a browse button and a text box for entering the filename.

The FileUpload control supports the following important properties.

- **FileBytes:** It returns the contents of uploaded file as a byte array
- **FileContent:** You can get the uploaded file contents as a stream.
- **FileName:** It provides the name of uploaded file.
- **HasFile:** It is a Boolean property that checks whether particular file is available or not.
- **PostedFile:** Gets the uploaded file wrapped in the `HttpPostedFile` object.

NOTES

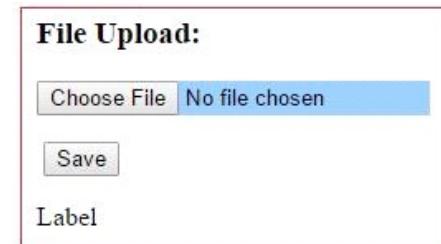
Example 11.2

```
using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void btnSave_Click(object sender, EventArgs e)
    {
        StringBuilder sb = new StringBuilder();
        if (FileUpload1.HasFile)
        {
            try
            {
                sb.AppendFormat(" Uploaded file: {0}",
FileUpload1.FileName);
                //save the file
                FileUpload1.SaveAs(@"C:\"
FileUpload1.FileName);
                //Showing the file information
                sb.Append("<br/> File Name: {0}" +
FileUpload1.PostedFile.FileName);
                sb.Append("<br/> File type: {0}" +
FileUpload1.PostedFile.ContentType);
                sb.Append("<br/> File length: {0}" +
FileUpload1.FileBytes.Length);
                Label1.Text = sb.ToString();
            }
        }
    }
}
```

```

        }
        catch (Exception ex)
        {
            sb.Append("<br/> Error <br/>");
            sb.Append(ex.Message);
            Label1.Text = sb.ToString();
        }
    }
}

```

NOTES**Output:****Uploading Large Files**

FileUpload control can be used for the file of size upto 4MB. Default settings cannot be used for the files larger than 4MB. For that, you need to change the setting. You need to configure the `httpRuntime maxRequestLength` and `httpRuntime requestLengthDiskThreshold` settings. This number is in KB.

```

<configuration>
    <system.web>
        <httpRuntime
            maxRequestLength="10240"
            requestLengthDiskThreshold="100" />
    </system.web>
</configuration>

```

The above web configuration file can upload up to 10MB file data.

Calendar Control

Calendar control provides you lots of property and events. Using these properties and events you can perform the following task with calendar control.

1. Select date.
2. Selecting a day, a week or a month.
3. Customize the calendar's appearance.

The Calendar control supports three important events which are tabulated below:

Event	Description
SelectionChanged	This event is fired when you select a day, a week or an entire month.
DayRender	This event is fired when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Calendar control supports SelectionMode property that allows you to select a single day, week, or entire month.

Example 11.3

```
using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Calendar1_SelectionChanged(object
sender, EventArgs e)
    {
        Label1.Text = "Todays date is: " +
Calendar1.TodaysDate.ToShortDateString();
        Label2.Text = "Your date of birth is: " +
Calendar1.SelectedDate.ToShortDateString();
    }
}
```

When you select a date, SelectionChanged event will be fired and displays the date in a label controls. In this example the date format is MM/DD/YYYY.

Output:



NOTES

NOTES**AdRotator control**

AdRotator control is used to display different advertisements randomly in a page. The list of advertisements is stored in either an XML file or in a database table. Lots of websites uses AdRotator control to display the advertisements on the web page.

To create an advertisement list, first add an XML file to your project.

Code for XML file

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
    <Ad>
        <ImageUrl><" /Images/logo1.png</ImageUrl>
        <NavigateUrl>http://www.TutorialRide.com</
        NavigateUrl>
        <AlternateText>Advertisement</AlternateText>
        <Impressions>100</Impressions>
        <Keyword>banner</Keyword>
    </Ad>
    <Ad>
        <ImageUrl><" /Images/logo2.png</ImageUrl>
        <NavigateUrl>http://www.TutorialRide.com</
        NavigateUrl>
        <AlternateText>Advertisement</AlternateText>
        <Impressions>100</Impressions>
        <Keyword>banner</Keyword>
    </Ad>
    <Ad>
        <ImageUrl><" /Images/logo3.png</ImageUrl>
        <NavigateUrl>http://www.CareerRide.com</
        NavigateUrl>
        <AlternateText>Advertisement</AlternateText>
        <Impressions>100</Impressions>
        <Keyword>banner</Keyword>
    </Ad>
    <Ad>
        <ImageUrl><" /Images/logo4.png</ImageUrl>
        <NavigateUrl>http://www.TutorialRide.com</
        NavigateUrl>
        <AlternateText>Advertisement</AlternateText>
        <Impressions>50</Impressions>
        <Keyword>banner</Keyword>
    </Ad>
```

```
</Ad>  
</Advertisements>
```

Basic Web Controls

In the given XML file ‘Images’ is the name of the folder, where we stored all the images to display. Now set the AdRotator control’s AdvertisementFile property. Set the path of the XML file that you have created to AdRotator control’s AdvertisementFile property.

Important Properties of AdRotator Control

ImageUrl: The URL of the image that will be displayed through AdRotator control.

- **NavigateUrl:** If the user clicks the banner or ad then the new page is opened according to given URL.
- **AlternateText:** It is used for displaying text instead of the picture if picture is not displayed. It is also used as a tooltip.
- **Impressions:** It is a number that sets how frequently an advertisement will appear.
- **Keyword:** It is used to filter ads or identifies a group of advertisement.

NOTES

MultiView control

MultiView control can be utilized when you need to make a selected page. As a rule, a web structure might be extremely long, and after that you can partition a long structure into different sub shapes. MultiView control is comprised of different view controls. You can put various ASP.NET controls inside view controls. One View control is shown at once and it is called as the dynamic view. View control does not work independently. It is constantly utilized with a Multiview control.

If you are working with Visual Studio 2010 or later, you can drag and drop a MultiView control onto the structure. You can drag and drop any number of View controls inside the MultiView control. The quantity of view controls is relies on the need of your application.

The MultiView control supports the following important properties.

- **ActiveViewIndex:** It is used to determine which view will be active or visible.
- **Views:** It provides the collection of View controls contained in the MultiView control.

To understand the Multiview control, first we will create a user interface as given below.

In the given example of Multiview control, we have taken three separate View control.

1. In First step we will design to capture Product details
2. In Second step we will design to capture Order details
3. Next we will show summary for confirmation.

Here, we have not used any database programming to save the data into the database. We will show only the confirmation page, that data has been saved.

NOTES
MultiViewControlDemo.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="RichControl.aspx.cs" Inherits="RichControl" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:MultiView ID="MultiView1" runat="server">
                <asp:View ID="View1" runat="server">
                    <table style="border:1px solid black">
                        <tr>
```

```
<td colspan="2">
    <h2>Step 1 - Product Details</h2>
</td>
</tr>
<tr>
    <td>Product ID</td>
    <td>
        <asp:TextBox ID="txtProductID" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>Product Name</td>
    <td>
        <asp:TextBox ID="txtProductName" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>Price/Unit</td>
    <td>
        <asp:TextBox ID="txtProductPrice" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td colspan="2" style="text-align:right">
        <asp:Button ID="btnStep2" runat="server" Text="Next >>" onclick="btnStep2_Click" />
    </td>
</tr>
</table>
</asp:View>
<asp:View ID="View2" runat="server">
    <table style="border:1px solid black">
        <tr>
            <td colspan="2">
                <h2>Step 2 - Order Details</h2>
            </td>
```

NOTES

NOTES

```
</tr>
<tr>
    <td>Order ID</td>
    <td>
        <asp:TextBox ID="txtOrderID"
            runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>Quantity</td>
    <td>
        <asp:TextBox ID="txtQuantity"
            runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
        <asp:Button ID="btnBackToStep1"
            runat="server" Text="<< Previous"
            onclick="btnBackToStep1_Click" />
    </td>
    <td style="text-align:right">
        <asp:Button ID="btnStep3"
            runat="server" Text="Next >>"
            onclick="btnGoToStep3_Click" />
    </td>
</tr>
</table>
</asp:View>
<asp:View ID="View3" runat="server">
    <table style="border:1px solid black">
        <tr>
            <td colspan="2"><h2>Step 3 - Summary</h2></td>
        </tr>
        <tr>
            <td colspan="2"><h3>Product Details</h3></td>
        </tr>
        <tr>
            <td>Product ID</td>
            <td>
```

```
<asp:Label ID="lblProductID"
    runat="server"></asp:Label>
</td>
</tr>
<tr>
    <td>Product Name</td>
    <td>
        <asp:Label ID="lblProductName"
            runat="server"></asp:Label>
    </td>
</tr>
<tr>
    <td>Price/Unit</td>
    <td>
        <asp:Label ID="lblPrice"
            runat="server"></asp:Label>
    </td>
</tr>
<tr>
    <td colspan="2"><h3>Order Details</h3></td>
</tr>
<tr>
    <td>Order ID</td>
    <td>
        <asp:Label ID="lblOrderID"
            runat="server"></asp:Label>
    </td>
</tr>
<tr>
    <td>Quantity</td>
    <td>
        <asp:Label ID="lblQuantity"
            runat="server"></asp:Label>
    </td>
</tr>
<tr>
    <td>
        <asp:Button ID="btnBackToStep2"
            runat="server" OnClick="btnBackToStep2_Click"
            style="height: 26px" Text="<<Previous" />
    </td>
```

NOTES

```
<td style="text-align:right">
    <asp:Button ID="btnSubmit" runat="server" Text="Submit >>" OnClick="btnSubmit_Click" />
</td>
</tr>
</table>
</asp:View>
</asp:MultiView>
</div>
</form>
</body>
</html>
```

NOTES

MultiViewControlDemo.aspx.cs file

```
using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (! IsPostBack)
        {
            MultiView1.ActiveViewIndex = 0;
        }
    }
    protected void btnStep2_Click(object sender, EventArgs e)
    {
        MultiView1.ActiveViewIndex = 1;
    }
    protected void btnBackToStep1_Click(object sender, EventArgs e)
    {
        MultiView1.ActiveViewIndex = 0;
    }
    protected void btnGoToStep3_Click(object sender, EventArgs e)
    {
        MultiView1.ActiveViewIndex = 2;
        lblProductID.Text = txtProductID.Text;
        lblProductName.Text = txtProductName.Text;
        lblPrice.Text = txtProductPrice.Text;
    }
}
```

```

        lblOrderID.Text = txtOrderID.Text;
        lblQuantity.Text = txtQuantity.Text;
    }
    protected void btnSubmit_Click(object sender, EventArgs
e)
    {
        Response.Redirect("SaveData.aspx");
    }
    protected void btnBackToStep2_Click(object sender,
EventArgs e)
    {
        MultiView1.ActiveViewIndex = 1;
    }
}

```

Basic Web Controls

NOTES

ActiveViewIndex property of MultiView control is zero based.

Wizard Control

Wizard control is same as MultiView control. The main difference is that, it has inbuilt navigation buttons. It enables to design a long form in such a way that you can work with multiple sub form. It reduces the overhead of developers to design multiple forms. This control provides built-in previous/next functionality.

The Wizard control can contains one or more WizardStep as child controls. Only one WizardStep is displayed at a time. StepType is the important property of the WizardStep control. This property determines the type of navigation buttons that will be displayed for that step. Following are the StepType:

- Start:
- Step:
- Finish:
- Complete:
- Auto:

You will get the following code on dragging the Wizard control on the web page from toolbox.

```

<asp:Wizard ID="Wizard1" runat="server" Height="75px"
Width="140px">
    <WizardSteps>
        <asp:WizardStep runat="server" title="Step 1">
        </asp:WizardStep>
        <asp:WizardStep runat="server" title="Step
2">
        </asp:WizardStep>

```

```
</WizardSteps>
</asp:Wizard>
```

You can put WizardStep according to application need.

NOTES

Important events of Wizard control are as follows:

- ActiveStepChanged:
- CancelButtonClick:
- FinishButtonClick:
- NextButtonClick:
- PreviousButtonClick:

Now we will create an application as we had done with MultiView control. We will create three different WizardStep in Wizard control.

- In First step, we will design to capture Product details
- In Second step, we will design to capture Order details
- Next, we will show summary for confirmation.



```
using System;
using System.Web.UI.WebControls;

public partial class WizardControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        protected void Wizard1_FinishButtonClick(object
sender, WizardNavigationEventArgs e)
        {
            Response.Redirect("SaveData.aspx");
        }
        protected void Wizard1_NextButtonClick(object sender,
WizardNavigationEventArgs e)
        {
            if (e.NextStepIndex == 2)
            {
                lblProductID.Text = txtProductID.Text;
                lblProductName.Text = txtProductName.Text;
                lblPrice.Text = txtProductPrice.Text;
                lblOrderID.Text = txtOrderID.Text;
                lblQuantity.Text = txtQuantity.Text;
            }
        }
    }
}
```

Register the event for Next button by using property window with event tab. In the given example, for going third step from second we have to set e.NextStepIndex == 2. Here StepIndex is zero based.

FinishButtonClick event performs the final WizardStep with a summary of the answers entered in the previous WizardStep controls.

NOTES

11.6 DATA BOUND CONTROLS

A data bound control is bound to a data source and generates its user interface by enumerating the items in the data source. ASP.NET provides a rich set of data bound server controls (see Figure 11.1). These controls are easy to use and provide a Rapid Application Development (RAD) for Web development.

You can categorize these controls into two groups:

1. Single item data bound control
2. Multi item data bound controls

NOTES

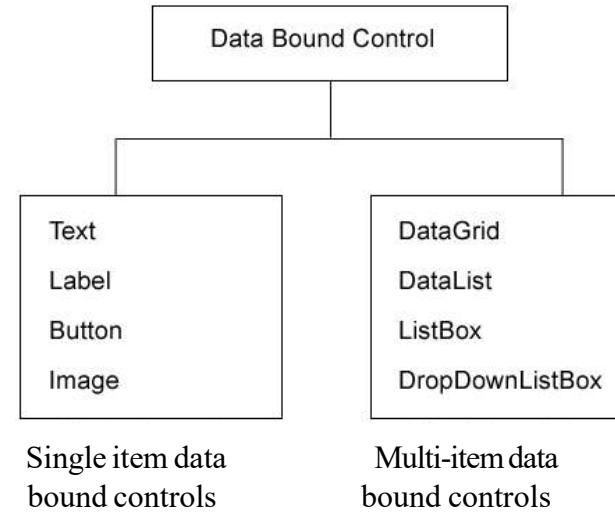


Fig. 11.1 Data Bound Control

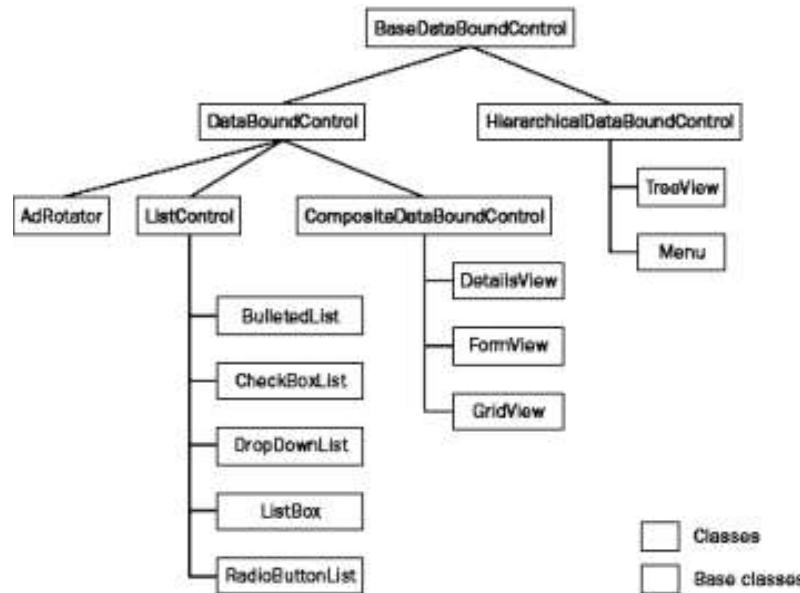
The single item data bound controls are used to display the value of a single item of a database table. These controls do not provide direct binding with the data source. Examples of single item data bound controls are Text Boxes, Buttons, Labels, Images, and so on.

The multi item data bound controls are used to display the entire or a partial table. These controls provide direct binding to the data source. You can use the `DataSource` property of these controls to bind a database table to these controls. Some examples of multi-item data bound controls are `DataGrid`, `ListBox`, `DataList` and `DropDownList`. Table 11.1 describes the various data bound controls.

Table 11.1 Data Bound Controls and their Description

Control	ASP.NET Code	Description
<code>DataGrid</code>	<code><asp:DataGrid></code>	Displays a database in a scrollable grid format and supports selection, adding, updating, sorting and paging.
<code>DataList</code>	<code><asp:DataList></code>	Displays data in templates and style format.
<code>ListBox</code>	<code><asp:ListBox></code>	Used to display data in a list format.
<code>DropDownList</code>	<code><asp:DropDownList></code>	Used to display ADO.NET data source data in a drop down combo box format.
<code>CheckBox</code>	<code><asp:CheckBox ></code>	Displays single check box, which can be connected to an item of the ADO.NET data source.
<code>CheckBoxList</code>	<code><asp:CheckBoxList></code>	Displays a list of check boxes that can be connected to the ADO.NET data source.
<code>Repeater</code>	<code><asp:Repeater></code>	Displays a template data bound list.
<code>TextBox</code>	<code><asp:TextBox></code>	Used to display a text box using ADO.NET Text property.

In ASP.NET, the class diagram of data bound controls is as shown in Figure 11.2.



NOTES

Fig. 11.2 Class Diagram of Data Bound Controls

All controls descend from the same base class—**BaseDataBoundControl**—regardless of the actual implementation or user interface characteristics. **BaseDataBoundControl** branches off into two more specific child classes—**DataBoundControl** and **HierarchicalDataBoundControl**.

The **DataBoundControl** Base Class

DataBoundControl is an abstract base class that defines the common characteristics of flat, non-hierarchical controls that use a data source. The class determines how a data bound control binds to a collection of data items or to a data source object.

Remarkably, in ASP.NET, all data bound controls—with the notable exceptions of **DownList**, **GridView** and **Repeater**—share a common base class.

You can group data bound controls into three categories: simple, composite and hierarchical controls. The following is a brief review of the properties and methods that all data bound controls share.

Properties of **DataBoundControl** Class

The **DataBoundControl** class inherits from **WebControl** and can, thus, have all the visual and style properties defined on the base class. Examples of visual properties include **BackColor**, **ForeColor**, **Font**, **BorderStyle** and the new **SkinID**. Table 11.2 shows the properties of the **DataBoundControl** class.

Table 11.2 Properties of DataBoundControl Class**NOTES**

Property	Description
DataMember	Selects the list of data that the control should bind to when the data source contains more than one list.
DataSource	Indicates the data source to bind to. The data source must be an object that implements the <code>IEnumerable</code> or <code>IListSource</code> interface.
DataSourceID	The ID of the data source objects is used to retrieve data.

Overview of Simple Data Bound Controls

The term ‘simple data bound control’ refers to Web controls that have a list-based user interface. The data source provides these controls with data items to populate a list of HTML markup elements. ASP.NET 2.0 has two types of simple data bound controls—the AdRotator control and List controls.

(i) The AdRotator Control

The `AdRotator` control displays advertising banner on a Web page that changes whenever the page refreshes. The control retrieves ad information from a separate XML file. For each ad, the XML file contains the path to a descriptive image, the URL (Uniform Resource Locator) to go to when the control is clicked and the frequency of the ad. For tracking purposes, the `AdRotator` control supports counters and updates them when an ad is clicked. You can associate each ad with its own counter.

(ii) The BulletedList Control

The `BulletedList` control is used to create a list of items formatted with bullets. Like all list controls, `BulletedList` allows you to specify individual items by defining a `ListItem` object for each desired entry.

The bulleted list generated by the code is bound to a Microsoft SQL Server data source and receives the `DataTable` returned by the specified query. The `DataTextField` property selects the column to show and `DisplayMode` sets the display mode—plain text, link buttons or hyperlinks.

Overview of Hierarchical Data Bound Controls

The `HierarchicalDataBoundControl` class is the foundation of hierarchical data bound controls, such as `TreeView` and `Menu`. The class acts as a logical container for controls that consume hierarchical data.

Tree-Based Data View

The `TreeView` control can bind to any data source object that supports the `HierarchicalDataSource` interface. By default, the `TreeView` binds XML nodes to its own nodes in a way that reflects the name of the node rather than a particular attribute or the inner text.

A `TreeNodeBinding` object defines the relationship between each data item and the node it is binding to.

Basic Web Controls

The Menu Control

The `Menu` control can be bound to any data source and also supports an explicit list of items for the simplest cases. An `ASP.NET Menu` object consists of menu items stored in a collection. `MenuItem` is the class that describes a single item, and `MenuItems` is the collection property that returns all the child items of a given menu. An example of a dynamic style is one that is assigned to the menu item currently under the mouse.

NOTES

Composite Data Bound Controls

A composite control manages a tree of child controls, and its output is obtained by merging the markup of the constituent components.

1. The `GridView`

The `GridView` is a major upgrade of the `ASP.NET 1.x DataGrid` control. It provides the same set of capabilities, plus a long list of extensions and improvements. The `DataGrid` has one big drawback: it requires you to write a lot of custom code, even to handle relatively simple operations, such as paging, sorting, editing or deleting data. The `GridView` control offers some new functionalities. These functionalities include features, such as automatic paging, sorting, selecting and editing. The `GridView` is also the only data control that can show more than one record at a time. An advantage of the `GridView` over the `DataGrid` is its support for code-free scenarios.

The `GridView` is an all-purpose grid control for displaying large tables of information. Each column represents a data source field and each row represents a record. It also exposes a new model for page developers to handle or cancel events.

`GridView` Control Fundamentals

1. Displays Data

The `GridView` renders its data items in an HTML table. Each data item is rendered in a distinct HTML table row.

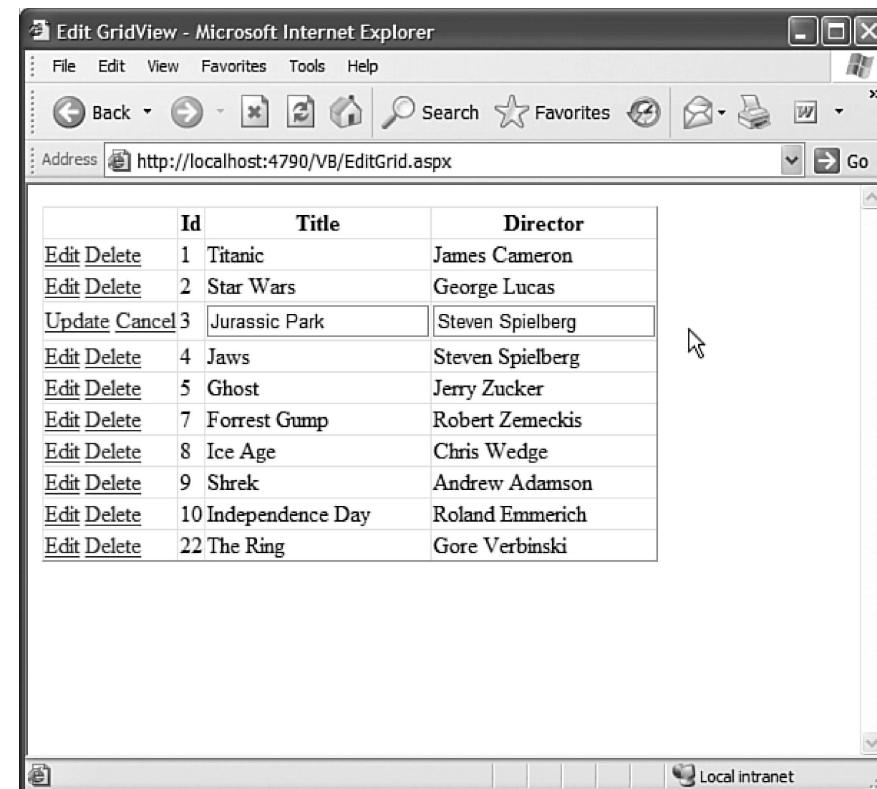
2. Paging through `GridView`

When working with a large number of database rows, it is useful to display the rows in different pages. You can enable paging with the `GridView` control by enabling its `AllowPaging` property.

(If you do not specify a value of `PageSize`, then the `GridView` defaults to displaying ten records per page.)

NOTES**3. Editing Data**

The `GridView` control also enables you to edit database data. The page illustrates how you can update and delete records in the `Movies` database table by using the `GridView` control.



The `GridView` control has both its `AutoGenerateEditButton` and `AutoGenerateDeleteButton` properties enabled. When these properties are enabled, Edit and Delete links are automatically rendered next to each row in the `GridView`. Tables 11.3 and 11.4 show the behavior and state properties, respectively.

Table 11.3 Behavior Properties

Behavior Property	Description
<code>AllowPaging</code>	Indicates whether the control supports paging.
<code>AllowSorting</code>	Indicates whether the control supports sorting.
<code>AutoGenerateColumns</code>	Indicates whether columns are automatically created for each field in the data source. The default is true.
<code>SortDirection</code>	A read only property that gets the direction of the column current sort.

Table 11.4 State Properties

Basic Web Controls

State Property	Description
DataKeyNames	Gets and sets an array that contains the names of the primary key fields for the currently displayed items.
EditIndex	Gets and sets the 0-based index that identifies the row currently rendered in edit mode.
PageCount	Gets the number of pages required to display the records of the data source.
PageIndex	Gets and sets the 0-based index that identifies the currently displayed page of data.
PageSize	Indicates the number of records to display on a page.
SelectedIndex	Gets and sets the 0-based index that identifies the row currently selected.
SelectedValue	Returns the explicit value of the key as stored in the DataKey object. Similar to SelectedDataKey.

NOTES

Formatting the GridView Controls

Table 11.5 shows that GridView control includes a rich set of formatting properties that you can use to modify its appearance.

Table 11.5 Formatting and Appearance Properties

Style Property	Description
AlternatingRowStyle	Defines the style properties for every other row displayed in the table.
EditRowStyle	Defines the style properties for the row being edited.
FooterStyle	Defines the style properties for the grid's footer.
HeaderStyle	Defines the style properties for the grid's header.
RowStyle	Defines the style properties for the rows displayed in the table.
Appearance Property	Description
CellPadding	Indicates the amount of space (in pixels) between the contents of a cell and the border.
CellSpacing	Indicates the amount of space (in pixels) between cells.
HorizontalAlign	Indicates the horizontal alignment of the control on the page.
ShowFooter	Indicates whether the footer row is displayed.
ShowHeader	Indicates whether the header row is displayed.

Supported Column Types in GridView Control

In all the sample code in the previous section, the GridView control was used to render automatically an HTML table that contains a list of data items. However, there is a problem with allowing the GridView to render its columns automatically. The result does not look very professional. The solution to all these problems is to specify explicitly the fields that a GridView displays. The GridView control supports the types of fields shown in Table 11.6.

Table 11.6 Field Types**NOTES**

Type	Description
BoundField	Default column type. Displays the value of a field as plain text.
ButtonField	Displays the value of a field as a command button. You can choose the link or the push button style.
CheckBoxField	Displays the value of a field as a check box. It is commonly used to render Boolean values.
CommandField	Enhanced version of ButtonField that represents special commands, such as Edit, Delete and Select.
HyperLinkField	Displays the value of a field as a hyperlink. When the hyperlink is clicked, the browser navigates to the specified URL. HyperLinkField accepts an array of data fields to build multi-parameter URLs.
ImageField	Displays the value of a field as an image. The image can come from a database or be specified through a URL.
TemplateField	Displays user-defined content for each item in the column. The template can contain any number of data fields combined with literals, images and other controls.

GridView Events

The GridView control includes a rich set of events that you can handle to customize the control's behavior and appearance. These events can be divided into three groups. First, the GridView control supports the set of events that are raised when the control displays its rows. Second, the GridView control includes the set of events that are raised when you are editing records. Third, the GridView control supports the events related to sorting, selecting and paging. Table 11.7 shows the properties of events.

Table 11.7 Properties of Events

Events	Description
PageIndexChanging, PageIndexChanged	Both events occur when one of the pager buttons is clicked. They fire before and after the grid control handles the paging operation, respectively.
RowCancelingEdit	Occurs when the Cancel button of a row in edit mode is clicked, but before the row exits edit mode.
RowDataBound	Occurs when a data row is bound to data.
RowDeleting, RowDeleted	Both events occur when a row's Delete button is clicked. They fire before and after the grid control deletes the row, respectively.
RowEditing	Occurs when a row's Edit button is clicked but before the control enters edit mode.
RowUpdating, RowUpdated	Both events occur when a row's Update button is clicked. They fire before and after the grid control updates the row, respectively.
SelectedIndexChanged, SelectedIndexChanging	Both events occur when a row's Select button is clicked. The two events occur before and after the grid control handles the select operation, respectively.

2. DataList CONTROL

DataList control is used to display a repeated list of items that are bound to the control. However, the DataList control adds a table around the data items by default.

The `DownList` control is used to display the data items in a repeating list. However, the support for selecting and editing the items is optional. The content and layout of list items in `DownList` is defined using templates.

Displaying Data with `DownList` Control

To display data with the `DownList` control, you must supply the control with an `ItemTemplate`. The contents of the `ItemTemplate` are rendered for each data item from the data source.

Moreover, several optional templates can be used to customize the appearance of the list. Table 11.8 describes those templates.

Table 11.8 Template Names and their Description

Template Name	Description
<code>ItemTemplate</code>	Defines the content and layout of items within the list. Required.
<code>AlternatingItemTemplate</code>	If defined, determines the content and layout of alternating items. If not defined, <code>ItemTemplate</code> is used.
<code>SeparatorTemplate</code>	If defined, is rendered between items (and alternating items). If not defined, a separator is not rendered.
<code>SelectedItemTemplate</code>	If defined, determines the content and layout of the selected item. If not defined, <code>ItemTemplate</code> (<code>AlternatingItemTemplate</code> , <code>SelectedItemTemplate</code>) is used.
<code>EditItemTemplate</code>	If defined, determines the content and layout of the item being edited. If not defined, <code>ItemTemplate</code> (<code>AlternatingItemTemplate</code> , <code>SelectedItemTemplate</code>) is used.
<code>HeaderTemplate</code>	If defined, determines the content and layout of the list header. If not defined, the header is not rendered.
<code>FooterTemplate</code>	If defined, determines the content and layout of the list footer. If not defined, the footer is not rendered.

NOTES

Formatting the `DownList` Control

The `DownList` control includes a rich set of properties that you can use to format the HTML rendered by the control. The following are its properties:

- **CssClass**—Enables you to associate a CSS (Cascading Style Sheet) class with the `DownList`.
- **AlternatingItemStyle**—Enables you to format every other row of the `DownList`.
- **EditItemStyle**—Enables you to format the `DownList` row selected for editing.
- **FooterStyle**—Enables you to format the footer row of the `DownList`.
- **HeaderStyle**—Enables you to format the header row of the `DownList`.

When formatting the `DownList`, you also need to work with the following properties:

- **GridLines**—Enables you to add rules around the cells in the `DownList`. Possible values are None, Horizontal, Vertical and Both.
- **ShowFooter**—Enables you to show or hide the footer row.

- **ShowHeader**—Enables you to show or hide the header row.
- **UseAccessibleHeader**—Enables you to render HTML <th> tags instead of <td> tags for the cells in the header row.

NOTES**Using the DataList Control**

To use this control, drag and drop the control from the toolbox in the design view of the Web form. The following screenshot displays a **DataList** control on a Web form:



The following list outlines the steps that you can follow to add a **DataList** control in a Web page and make it working:

1. Drag and drop a **DataList** control in the Web form from the toolbox.
2. Set the **DataSourceID** property of the control to the data source that you will use to bind data to the control.
3. Open the .aspx file, declare the element and define the fields as per your requirements.
4. Bind the data through the `Eval()` method to display data in these defined fields of the control.

You can bind data to the **DataList** control in two different ways, i.e., using the **DataSourceID** and the **DataSource** properties. You can use inbuilt features, such as selecting and updating data when using the **DataSourceID** property.

3. Repeater Control

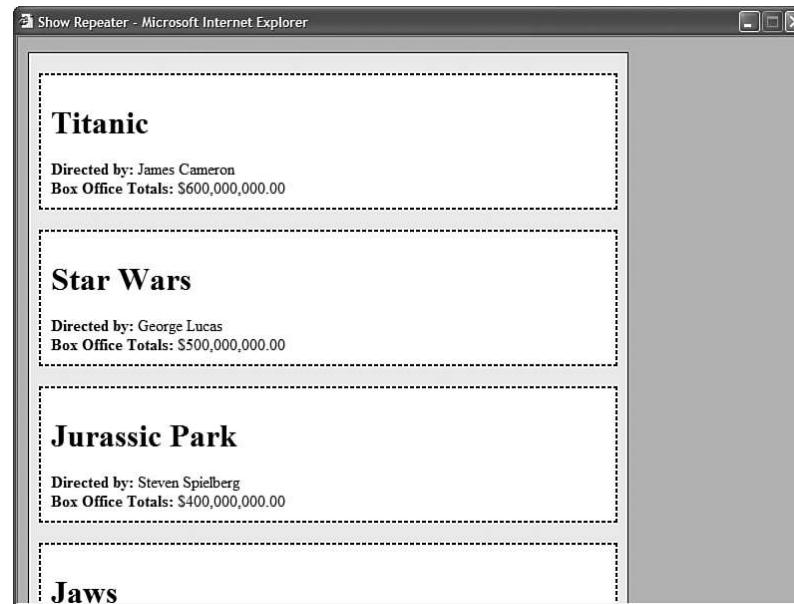
In **ASP.NET**, **Repeater Control** provides the functionality as a data bound control and displays the retrieved data in a repeating list form. It is a basic data bound container control that allows authors to customize the display of multiple rows of data using HTML tags and CSS styles.

The **Repeater** control provides you with the maximum amount of flexibility in rendering a set of database records. You can format the output of the **Repeater** control in any way that you want.

Displaying Data with Repeater Control

Basic Web Controls

To display data with the Repeater control, you must create an ItemTemplate.



NOTES

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
<style type="text/css">
html
{
background-color:silver;
}
.content
{
width:600px;
border:solid 1px black;
background-color:#eeeeee;
}
.movies
{
margin:20px 10px;
padding:10px;
border:dashed 2px black;
```

NOTES

```
background-color:white;
}

</style>

<title>Show Repeater</title>
</head>
<body>
<form id="form1" runat="server">
<div class="content">
<asp:Repeater id="rptMovies" DataSourceID="srcMovies"
Runat="server">
<ItemTemplate>
<div class="movies">
<h1><%#Eval("Title") %></h1>
<b>Directed by:</b> <%# Eval("Director") %>
<br />
<b>Box Office Totals : </b> <%#
Eval("BoxOfficeTotals", "{0:c}") %>
</div>
</ItemTemplate>
</asp:Repeater>
<asp:SqlDataSource id="srcMovies"
ConnectionString="<%$ConnectionStrings: Movies %>" 
SelectCommand="SELECT Title, Director, BoxOfficeTotals FROM
Movies" Runat="server" />
</div>
</form>
</body>
</html>
```

The content and layout of list items is defined using templates shown in Table 11.9.

Table 11.9 Template Names

Basic Web Controls

Template Name	Description
ItemTemplate	Defines the content and layout of items within the list.
AlternatingItemTemplate	If defined, the alternatingItemTemplate determines the content and layout of alternating items. If not defined, ItemTemplate is used.
SeparatorTemplate	If defined, SeparatorTemplate is put across between items (and alternating items). If not defined, a separator is not rendered.
SelectedItemTemplate	If defined, SelectedItemTemplate determines the content and layout of the selected item. If not defined, ItemTemplate (AlternatingItemTemplate , SelectedItemTemplate) is used.
EditItemTemplate	If defined, EditItemTemplate determines the content and layout of the item being edited. If not defined, ItemTemplate (AlternatingItemTemplate , SelectedItemTemplate) is used.
HeaderTemplate	If defined, HeaderTemplate determines the content and layout of the list header. If not defined, the header is not rendered.
FooterTemplate	If defined, FooterTemplate determines the content and layout of the list footer. If not defined, the footer is not rendered.

NOTES

Templates define the HTML elements and controls that should be displayed for an item, as well as the layout of these elements. Style formatting, such as font, color and border attributes are set via styles. Each template has its own style property. For example, styles for the **EditItemTemplate** are set through the **EditItemStyle** property.

```
<asp:DataList id="dlEmployee" runat="server">  
    <HeaderTemplate>  
        ...  
    </HeaderTemplate>  
    <ItemTemplate>  
        ...  
    </ItemTemplate>  
    <AlternatingItemTemplate>  
        ...  
    </AlternatingItemTemplate>  
    <FooterTemplate>  
        ...  
    </FooterTemplate>  
</asp:DataList>
```

Repeater Control Events

The Repeater control supports the following events:

- **DataBinding**—It is raised when the Repeater control is bound to its data source.
- **ItemCommand**—It is raised when a control contained in the Repeater control raises an event.

- **ItemCreated**—It is raised when each Repeater item is created.
- **ItemDataBound**—Raised when each Repeater item is bound.

4. DetailsView CONTROL

NOTES

The DetailsView control is complementary to the GridView control. In ASP.NET, it enables working with a single record or row from an associated data source. The default view of this control is vertical with each column of the record displayed on a line of its own. It is used for updating and inserting new records.

The DetailsView control rides on the data source capabilities to update insert or delete rows. It renders a single data item at a time even when the data source exposes more items. If the data object represents the `ICollection` interface or the underlying data source exposes the paging operation, the DetailsView can page over the data, provided the `AllowPaging` property is set to true. Users can also navigate between rows of data using this control.

The process of binding the DetailsView control to a data source is very simple. The `DataSourceID` property can be set declaratively or programmatically. If the `AutoGenerateEditButton` property is set to true, it enables editing operations. The Edit mode can be invoked pressing this button.

The DetailsView control can be configured to display a Delete and Insert button also. The `AutoGenerateInsertButton`, when set to true, generates a new button. The new button invokes the insert mode. Using this collection, the data source control can locate the unique row to be updated.

The DetailsView control user interface can be customized using `Style` properties, such as `HeaderRowStyle`, `RowStyle`, `AlternatingRowStyle`, and so on. `EmptyDataTemplate`, `HeaderTemplate`, `FooterTemplate` and `PagerTemplate` are some of the templates available for this control.

A number of events can be used to customize the code including pre and post insert, update and delete data source events. This includes `ItemCreated` and `ItemCommand` events. There is no selected event in the DetailsView control as the current event is treated as the selected event. The Control can also be used for rendering mobile devices. Table 11.10 shows the layout properties of DetailsView control.

Table 11.10 Layout Properties of DetailsView Control

Basic Web Controls

Property	Description
AllowEdit	Indicates whether the Edit button should be displayed in the command bar. False by default.
AllowInsert	Indicates whether the Insert button should be displayed in the command bar. False by default.
AllowPaging	Indicates whether the control should display the navigation bar. False by default.
CurrentMode	Indicates the current working mode of the control. Feasible values come from the <code>DetailsViewMode</code> enum type: Read Only, Edit and Insert.
EmptyText	The text to display should the data source be empty.
FooterText	The text to display on the footer of the control.
HeaderText	The text to display atop the control.
ViewTemplateUrl	Indicates the URL for the ASCX user control to use to provide a customized view of the record.

NOTES

5. FormView CONTROL

FormView is a new data bound control that is a templated version of DetailsView control. FormView Control is similar to DetailsView control. Both controls enable showing a single record from data source. The main difference between FormView and DetailsView controls is that FormView control allows using of templates for displaying a record instead of row fields used in DetailsView. Unlike DetailsView, FormView control does not have predefined data layout. The developer needs to create a template first and specify controls and formatting to display single record from data source. Thus, the developer has full control over layout and it is pretty easy to add validation capabilities.

Its properties, such as BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, and Height, are implemented through style properties of `<table>` tag.

Table 11.11 some important properties that are very useful.

Table 11.11 Templates and Methods of FormView Control

Templates of the FormView Control	
EditItemTemplate	Used when a record is being edited.
InsertItemTemplate	Used when a record is being created.
ItemTemplate	Used to render the record to display only.
Methods of the FormView Control	
ChangeMode	Read Only/Insert/Edit. Change the working mode of the control from the current to the defined <code>FormViewMode</code> type.
InsertItem	Used to insert the record into database. This method must be called when the DetailsView control is in insert mode.
UpdateItem	Used to update the current record into database. This method must be called when DetailsView control is in edit mode.
DeleteItem	Used to delete the current record from database.

NOTES

11.7 CUSTOM CONTROLS

ASP.NET allows the users to create controls. Custom controls are the controls that are not included in the .Net framework but are created by a user or a third party software vendor. These user defined controls are categorized into:

1. User controls
2. Custom controls

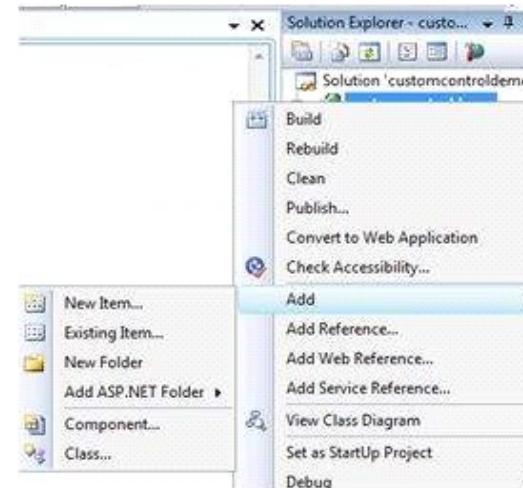
User Controls

These are like page or web forms in ASP.NET that are used for specific purpose and can be used by many other pages. These controls are derived from the `System.Web.UI.UserControl` class. Following are the characteristics of these controls:

- They have an `.ascx` extension.
- They may not contain any `<html>`, `<body>`, or `<form>` tags.
- They have a Control directive instead of a Page directive.

In order to understand the concept, we will create a simple user control that is used as footer for the web pages. You can use the following steps to create the user control.

1. Create a new web application.
2. Right click on the project folder on the solution explorer and choose Add New Item.



3. Select Web User Control from the Add New Item dialog box and name it `footer.ascx`. Initially, the `footer.ascx` contains only a Control directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="footer.ascx.cs"
Inherits="customcontroldemo.footer" %>
```

4. Add the following code to the file:

Basic Web Controls

```
<table>
<tr>
 Copyright ©2010 TutorialPoints Ltd.</td> </tr>  <tr>  Location: Hyderabad, A.P </td> </tr> </table> | |
```

NOTES

You must have to add the Register directive and an instance of the user control to the page to add the user control. The code given below shows the content file.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="customcontroldemo.
_Default" %>

<%@ Register Src("~/footer.ascx" TagName="footer"
TagPrefix="Tfooter" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional/
/EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">
    <title>
        Untitled Page
    </title>
</head>

<body>

<form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server"
Text="Welcome to ASP.Net Tutorials "></asp:Label>
```

```
<br /> <br />
<asp:Button ID="Button1" runat="server"
onclick="Button1_Click" Text="Copyright Info" />
```

NOTES

```
</div>
<Tfooter:footer ID="footer1" runat="server" />
</form>

</body>
</html>
```

When the above code gets executed, the page shows the footer and this control could be used in all the pages of your website.

Observe the following:

- (1)** The Register directive specifies a tag name as well as tag prefix for the control.

```
<%@ Register Src("~/footer.ascx" TagName="footer"
TagPrefix="Tfooter" %>
```

- (2)** The following tag name and prefix should be used while adding the user control on the page:

```
<Tfooter:footer ID="footer1" runat="server" />
```

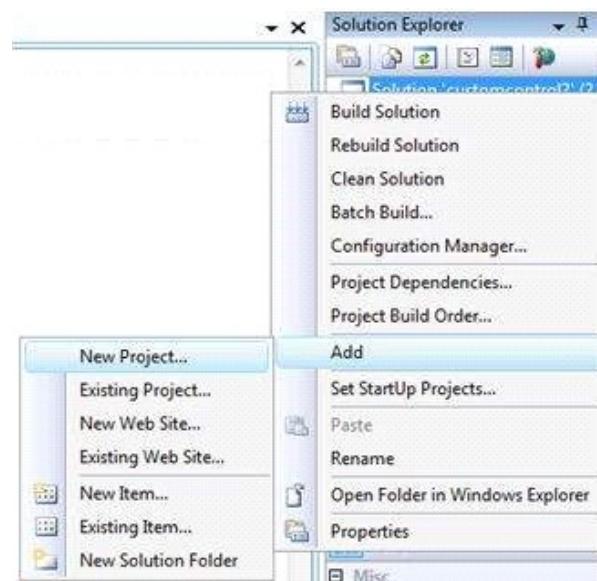
Custom Controls

Custom controls are deployed as individual assemblies. They are compiled into a Dynamic Link Library (DLL) and are used as any other ASP.NET server control. These controls can be created using any of the following ways:

1. By deriving a custom control from an existing control
2. By composing a new custom control combining two or more existing controls.
3. By deriving from the base control class.

Let us consider an example for creating a custom control that simply renders a text message on the browser. Following are the steps.

1. Create a new website.
2. Right click the solution (not the project) at the top of the tree in the Solution Explorer.

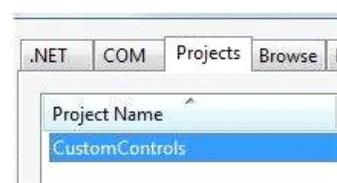


NOTES

3. Select ASP.NET Server Control from the project templates.



4. The above step adds a new project and creates a complete custom control to the solution, called ServerControl1. In this example, let us name the project CustomControls. To use this control, this must be added as a reference to the web site before registering it on a page. To add a reference to the existing project, right click on the project (not the solution), and click Add Reference.
5. Select the CustomControls project from the Projects tab of the Add Reference dialog box. The Solution Explorer should show the reference.



To use the control on a page, add the Register directive just below the @Page directive:

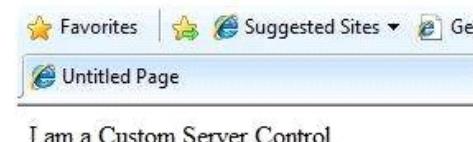
```
<%@ Register Assembly="CustomControls" Namespace=
"CustomControls" TagPrefix="ccs" %>
```

NOTES

Further, you can use the control, similar to any other controls.

```
<form id="form1" runat="server">
    <div>
        <ccs:ServerControl1 runat="server" Text = "I am a
Custom Server Control" />
    </div>
</form>
```

When executed, the Text property of the control is rendered on the browser as shown below:



Working with Custom Controls

In the previous example, the value for the Text property of the custom control was set. ASP.NET adds this property by default, when the control was created. The following code behind file of the control reveals this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControls
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server>{/}
{0}:ServerControl1 >")]
}

public class ServerControl1 : WebControl
{
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Localizable(true)]
```

```

public string Text
{
    get
    {
        String s = (String)ViewState["Text"];
        return ((s == null) ? "[" + this.ID + "]" :
s);
    }

    set
    {
        ViewState["Text"] = value;
    }
}

protected override void
RenderContents(HtmlTextWriter output)
{
    output.Write(Text);
}
}

```

NOTES

The above code is automatically generated for a custom control. Events and methods could be added to the custom control class.

Example 11.4

Let us consider the previous custom control named SeverControl1. Here, we add a method named checkpalindrome to check for palindromes. The code is given below:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControls

```

NOTES

```

{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></
{0}:ServerControl1 >")]
}

public class ServerControl1 : WebControl
{
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Localizable(true)]

    public string Text
    {
        get
        {
            String s = (String)ViewState["Text"];
            return ((s == null) ? "[" + this.ID + "]": s);
        }

        set
        {
            ViewState["Text"] = value;
        }
    }

    protected override void
RenderContents(HtmlTextWriter output)
{
    if (this.checkpanlindrome())
    {
        output.Write("This is a palindrome: <br />");
        output.Write("<FONT size=5 color=Blue>");
        output.Write("<B>");
        output.Write(Text);
        output.Write("</B>");
        output.Write("</FONT>");
    }
    else
}
}

```

```
{  
    output.Write("This is not a palindrome: <br />");  
    output.Write("<FONT size=5 color=red>");  
    output.Write("<B>");  
    output.Write(Text);  
    output.Write("</B>");  
    output.Write("</FONT>");  
}  
}  
  
protected bool checkpanlindrome()  
{  
    if (this.Text != null)  
    {  
        String str = this.Text;  
        String strtoupper = Text.ToUpper();  
        char[] rev = strtoupper.ToCharArray();  
        Array.Reverse(rev);  
        String strrev = new String(rev);  
  
        if (strtoupper == strrev)  
        {  
            return true;  
        }  
        else  
        {  
            return false;  
        }  
    }  
    else  
    {  
        return false;  
    }  
}  
}  
}
```

NOTES

When you change the code for the control, you must build the solution by clicking Build → Build Solution, so that the changes are reflected in your project. Add a text box and a button control to the page, so that the user can provide a text, it is checked for palindrome, when the button is clicked.

NOTES

```

<form id="form1" runat="server">
    <div>
        Enter a word:
        <br />
        <asp:TextBox ID="TextBox1" runat="server"
            style="width:198px"> </asp:TextBox>

        <br /> <br />

        <asp:Button ID="Button1" runat="server"
            onclick="Button1_Click" Text="Check Palindrome"
            style="width:132px" />

        <br /> <br />

        <ccs:ServerControl1 ID="ServerControl11"
            runat="server" Text = "" />
    </div>
</form>

```

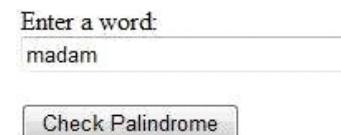
The Click event handler for the button simply copies the text from the text box to the Text property of the custom control.

```

protected void Button1_Click(object sender, EventArgs e)
{
    this.ServerControl11.Text = this.TextBox1.Text;
}

```

When executed, the control successfully checks palindromes.



Observe the following:

- (1) When you add a reference to the custom control, it is added to the toolbox and you can directly use it from the toolbox similar to any other control.



- (2) The RenderContents method of the custom control class is overridden here, as you can add your own methods and events.
- (3) The RenderContents method takes a parameter of HtmlTextWriter type, which is responsible for rendering on the browser.

NOTES**Check Your Progress**

3. Which class is used to inherit the validation control classes?
4. What are the control that comes under the rich controls?
5. What are the three ways of creating the custom control?

11.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A special tag is used to define ASP.Net web controls which starts with “`<asp:`” prefix followed by the corresponding control class name. For example, the tag used for TextBox controls is `<asp:TextBox>` and the tag used for Label controls is “`<asp:Label>`”.
2. List controls are derived from the class `System.Web.UI.WebControls.ListControl`.
3. The validation control classes are inherited from the `BaseValidator` class.
4. Following are the control that comes under the rich controls:
 - (i) FileUpload control
 - (ii) Calendar control
 - (iii) AdRotator control
 - (iv) MultiView control
 - (v) Wizard control
5. Custom controls can be created using any of the following ways:
 - (i) By deriving a custom control from an existing control
 - (ii) By composing a new custom control combining two or more existing controls.
 - (iii) By deriving from the base control class.

11.9 SUMMARY

- Controls are the basic building blocks of the Graphical User Interface (GUI) that enables users to interact with the user by allowing them to enter data, make selections to indicate their preference and to display data.

NOTES

- ASP.NET server controls are the primary controls used in ASP.NET and are also known as web controls.
- A special tag is used to define ASP.Net web controls which starts with “`<asp:`” prefix followed by the corresponding control class name.
- ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.
- A Tree view control comes under navigation controls. Other Navigation controls are: Menu control and SiteMapPath control.
- The collection of items is displayed through the List control that enables to display simple lists of options.
- ASP.NET validation controls perform validation on both, the client (or browser) and the server to ensure that useless, unauthenticated, or contradictory data did not get stored.
- The validation control classes are inherited from the BaseValidator class.
- Rich controls contain rich functionality and are built with multiple HTML elements.
- AdRotator control is used to display different advertisements randomly in a page. The list of advertisements is stored in either an XML file or in a database table.
- Wizard control is same as MultiView control. The main difference is that, it has inbuilt navigation buttons.
- A data bound control is bound to a data source and generates its user interface by enumerating the items in the data source.
- Custom controls are the controls that are not included in the .Net framework but are created by a user or a third party software vendor.

11.10 KEY WORDS

- **Server Controls:** These are classes in the .NET Framework that represent visual elements on a Web Form.
- **Validation:** The method of checking and evaluating that the user has entered the correct values in input fields.
- **TreeView Control:** It is used to display hierarchical data in the TreeView format.

11.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What are the different categories of web controls?
2. Discuss the common properties of ListControl class.
3. What are the different types of validation controls?
4. List the members of BaseValidator class.
5. Write a note on MultiView and wizard control.
6. What is the difference between DataList and Repeater control?

NOTES

Long Answer Questions

1. Write and explain some of the properties of server controls.
2. Write and explain some of the methods of server controls.
3. Explain the various types of validation controls.
4. Explain any two rich controls.
5. Describe the properties of the DetailsView control.
6. Explain the process of creating the custom controls.

11.12 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 12 OVERVIEW OF AJAX

NOTES

Structure

- 12.0 Introduction
 - 12.1 Objectives
 - 12.2 The AJAX Vision
 - 12.3 Server-side AJAX versus Client-side AJAX
 - 12.4 The UpdatePanel Control
 - 12.5 The Timer Control
 - 12.6 The UpdateProgress Control
 - 12.7 The ASP.NET AJAX Control Toolkit
 - 12.8 Answers to Check Your Progress Questions
 - 12.9 Summary
 - 12.10 Key Words
 - 12.11 Self Assessment Questions and Exercises
 - 12.12 Further Readings
-

12.0 INTRODUCTION

The biggest challenge faced by Web application developers is creating a fast and responsive user interface. So far, you have learned to build Web pages that use the postback model. With the postback model, pages are perpetually sent back to the Web Server and regenerated.

Recently, a new generation of Web applications has evolved that behave more like Windows applications than conventional Web pages, and these applications refresh themselves quickly. This new breed of Web applications uses a set of design patterns and technologies known as **AJAX** (Asynchronous JavaScript and XML). **AJAX** is a programming shorthand for a set of techniques that create more responsive, dynamic pages. One of the features of **AJAX** is the ability to refresh part of the page while leaving the rest intact.

This chapter explores the features of **AJAX** where you will learn how **AJAX** works and how you can use it to create rich, responsive Web pages. Instead of delving into the intricacies of **AJAX**, you will explore the **AJAX** features of **ASP.NET**. You will also learn about **AJAX** control: the **UpdatePanel** control. This control enables you to easily furnish existing **ASP.NET** applications with **AJAX** functionality. You will use the **UpdatePanel** control to update content in a page without posting the entire page back to the Web Server. You will also learn about the two controls that support the **UpdatePanel** control: the **Timer** control and the **UpdateProgress** control.

12.1 OBJECTIVES

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

- Explain the features of AJAX vision
- Update the contents of a page using the UpdatePanel control
- Refresh an UpdatePanel at intervals using the ASP.NET AJAX Timer control
- Display a progress indicator using the UpdateProgress control

NOTES

12.2 THE AJAX VISION

AJAX is definitely the buzzword in the Web application world at the moment. AJAX is a cross-platform combination of technologies that can be used to make your Web pages fast, rich and responsive. It is a relatively new Internet technology that grew out of combining JavaScript and XML.

Since its introduction, AJAX has produced radical changes in how Web applications are developed. It indicates the ability to develop applications that make use of the XMLHttpRequest object. This object could be used to retrieve data from the server asynchronously. The XMLHttpRequest object later paved the way for the birth of AJAX. It was this concept (usage of XMLHttpRequest object to perform asynchronous operations) that was named AJAX by Jesse James Garrett of Adaptive Path in early 2005.

AJAX made its presence felt within development communities worldwide in late 2005. It was Google who first led the drive to make AJAX known to communities worldwide by announcing the first public implementation in Google Suggest. Since then, AJAX applications gained popularity after Google released a significant number of applications, such as Google Docs, Google Maps and Gmail. With Microsoft taking AJAX to new heights and coming up with new releases, AJAX is set to become the technology of choice for building Web applications in the years to come. By using AJAX, you can cut down on network load and bandwidth usage and retrieve only the data you require.

Virtually, every control in ASP.NET uses a postback to the server in that it is executed on the server and not on the browser. All the form information from an ASP.NET page has the ability to post information to the server and return it to the same page. The problem arises when everything in a page is returned to refresh just a little information. For example, text takes up very little Internet space and is pretty fast to send all by itself. However, if you change just the text, the whole page along with the updated text gets refreshed and that slows the process.

The key feature of AJAX can be seen when the client requests new information. Instead of sending a whole new page, AJAX sends only the required new information. This means that all the buttons, text, graphics and other objects that may be on a page do not have to be reloaded into your browser.

NOTES

Today, if you are going to build an application, you have the option of creating a thick-client or a thin-client application. A thick-client application is usually a compiled executable code that end-users can run in their own environment. Generally, the technology to build this type of application is the Windows Forms technology, or MFC (Microsoft Foundation Class) in the C++ world.

A thin-client application is usually one that has its processing and rendering controlled at a single point and the outcome of the view is sent to a browser so that the client can view it. This type of technology requires that the client be connected to the Internet or an Intranet of some kind.

Each type of application has its pros and cons. The thick-client style of application is touted as more fluid and more responsive to an end-user's actions. In a Web-based application, the complaint has been for many years that every action by an end-user takes numerous seconds and results in a jerky page refresh. In turn, the problem with a thick-client style of application has always been that the application sits on the end user's machine and any patches or updates to the application require you to somehow upgrade each and every machine upon which the application sits. In contrast, the thin-client application, or the Web application architecture includes only one instance of the application. The application in this case is installed on a Web Server and any updates that need to occur happen only to this instance. End-users who call the application through their browsers always get the latest and largest version of the application.

ASP.NET AJAX further eliminates any of the negatives that would have stopped you from building an application on the Web. ASP.NET AJAX makes your Web applications seem more efficient than ever before. AJAX-enabled applications are responsive and give the end-user immediate feedback and guidance through the workflows that you provide. The power of this alone makes the study of this new technology and its incorporation into your projects a major importance.

Before you really get started with AJAX, it is important to understand its capabilities and limitations. Only then will you know how to fit it into your Web applications.

Advantages of AJAX

- **Responsiveness:** The key benefit of AJAX is responsiveness. An AJAX application, when done properly, provides a better experience for the user. Even if the user cannot do anything new, this improved experience can make your Web application seem more modern and sophisticated.

AJAX can also provide genuinely new features that are not possible in traditional Web pages. For example, AJAX pages often use JavaScript code that reacts to client-side events. These events occur frequently; so it is not practical to deal with them using the postback model.

- **Usability:** AJAX provides a way for the user to interact with a Website without refreshing and without any browser plug-ins.

- **Bandwidth Saving:** The next advantage to AJAX is bandwidth saving. When an application uses AJAX to communicate with the server, only the required information is transmitted to/from the server instead of a full transmission.

Disadvantages of AJAX

- **Security:** One big disadvantage of using AJAX is security. Often, developers do not put checks on the data coming into the server—they assume that it is from their own Website. Unfortunately, this is subject to injection attacks.
- **Encrypted Environments:** Another disadvantage is that AJAX does not play well in encrypted environments. AJAX relies on plain text transmission and so having the server-side program deal with it presents a large problem.
- **Complexity:** Writing the JavaScript code needed to implement an AJAX application is difficult.
- **Browser Support:** Another challenge to using AJAX is browser support. The technology that supports AJAX has existed for several years, but it is only now found consistently in all major browsers. If you use the AJAX features that ASP.NET provides, they will work in Internet Explorer 5 and newer version of Netscape 7, Opera 7.6, Safari 1.2, and Firefox 1.0
- **Time:** Building an AJAX application from scratch is actually fairly easy, but it does take longer than building one that uses standard postbacks. Moreover, the developers require a fairly good knowledge of JavaScript to be able to make it usable.

Table 12.1 lists the advantages and disadvantages of AJAX techniques.

Table 12.1 Advantages and Disadvantages of AJAX Techniques

Technique	Advantages	Disadvantages
XMLHttpRequest	1. Can make requests to pages not set up for AJAX. 2. Can set/get all HTTP headers. 3. Can make HTTP requests using any type (GET and POST). 4. Supports full control over POST requests, allowing for any type of data encoding.	1. Requests ActiveX to be enabled in IE 5 and 6. 2. Is only available in newer versions of Opera and Safari. 3. Has small implementation differences between browsers.
Iframe	1. Can make POST and GET HTTP requests. 2. Supports all modern browsers. 3. Supports asynchronous file uploads.	1. Prohibits synchronous requests. 2. Server pages must be designed to work with Iframe requests. 3. It has implementation differences between browsers. 4. Can leave extra entries in browser history. 5. All request data is URL-encoded, increasing request size.
Cookies	1. Supports the largest number of browsers. 2. Few implementation differences between browsers.	1. Prohibits no synchronous requests. 2. Does not work with large requests/results. 3. Requires server pages to be designed to work with cookie requests. 4. Requires polling on the client. 5. Can make only GET HTTP requests.

NOTES

NOTES

12.3 SERVER-SIDE AJAX VERSUS CLIENT-SIDE AJAX

ASP.NET AJAX is now just part of the ASP.NET Framework. When you create a new Web application, you do not have to create a separate type of ASP.NET application. Instead, all ASP.NET applications that you create are now AJAX-enabled.

In order to make things simple, the AJAX frameworks can be classified into two major parts:

1. Server-side framework
2. Client-side framework

Each of the above-listed framework is discussed in detail in the following sections.

1. Server-Side Framework

Server-side frameworks as the name suggests are used on the server-side. It is not actually JavaScript that is used in server-side framework, in fact you use the same language API (Application Programming Interface) is provided by the framework.

Developers can use this framework and install specific functions from the library so that it could be interpreted and rendered effectively as an AJAX-based application. This type of framework is usually used by developers who are not skilled in JavaScript. The most popular among this type of frameworks is the Google Web Toolkit or GWT. Usually, these frameworks could be implemented in different browsers.

Google Web Toolkit (GWT) is such a server-side framework which is used as a normal Java Swing kind of programming on the server-side. The GWT provides a tool for deploying your Web application having such pages. During the deployment of the Web application, the GWT tools help in translating the Java syntax into the corresponding AJAX functionality to be brought on the client-side.

Advantage of Server-Side Framework

The advantage of using a server-side framework is that you use the existing language you are working in. You do not have to dwell in the wild world of JavaScript. It is easier to use a server-side framework rather than to learn JavaScript. Other server-side frameworks are: **Yahoo Toolkit and DWR** (Direct Web Reporting).

2. Client-Side Framework

On the other hand, client-side framework is invoked within the user's browser. Most coding on AJAX today revolves around this type of framework. Probably, the best side of this framework is the ability to build widgets so that it could be integrated into a desktop environment.

In a large application, much more functionality is required. The following are some of these functionalities:

- **Cross Browser Compatibility:** When you code, it is always a necessity that the code works independent of the browser type requesting it.
- **Multiple Request Handling:** Multiple asynchronous requests have to be handled properly when doing the callbacks.
- **Graceful Degradation:** You do not want your code to fall flat in case the browser does not support the XMLHttpRequest object.
- **Exception and Error Handling:** You should be able to catch the exceptions in your code instead of displaying it to the user.

NOTES

To do all these by yourself is very tedious and requires a lot of JavaScript expertise. Hence, to avoid these, we have client side frameworks. They provide packages and APIs which help you to implement all of the above functionalities and also provide numerous extra features which currently can be either found in free JavaScript code snippets over the Internet.

The extra features include widgets or even an IDE (Integrated Development Environment). Widgets are small interfaces or components which can be incorporated in an application easily through the client-side frameworks.

These widgets might compose of the following:

1. Auto Completing Combo Box
2. A Menu System
3. An Accordion
4. A Tabbed Pane Windows
5. Effects like Fade, Zoom In/Out

Advantage of Client-Side Framework

An important advantage of a client-side framework over a server-side framework is that you have complete flexibility in your hands. You can mix and match the widget features and at the same time provide extra flexibility. Some frameworks like **TIBCO GI** even provide a GUI (Graphical User Interface) to build your Web pages with AJAX components in a complete drag and drop style. The examples of a client-side framework are **Prototype**, **DOJO Toolkit**, **Rico**, **Script.aculo.us**, and so on.

ASP.NET AJAX's Server Side Controls

The controls are focused on allowing you to AJAX-enable your ASP.NET applications. They are enabling controls. Table 12.2 lists the properties of AJAX server-side controls.

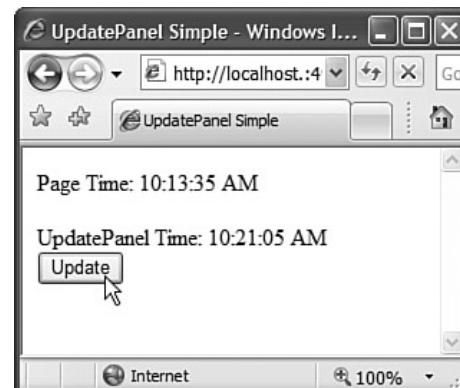
Table 12.2 Properties of AJAX Server-Side Controls**NOTES**

AJAX Server Side Controls	Description
ScriptManager	A component control that manages the marshalling of messages to the AJAX-enabled server for the parts of the page requiring partial updates. Every ASP.NET page will require a ScriptManager control in order to work. It is important to note that you can have only a single ScriptManager control on a page.
ScriptManagerProxy	A component control that acts as a ScriptManager control for a content page. The ScriptManagerProxy control, which sits on the content page, works in conjunction with a required ScriptManager control that resides on the master page.
Timer	The Timer control will execute client-side events at specific intervals and allows specific parts of your page to update or refresh at these moments.
UpdatePanel	A container control that allows you to define specific areas of the page that are enabled to work with the ScriptManager. These areas can then, in turn, make the partial page postbacks and update themselves outside the normal ASP.NET page postback process.
UpdateProgress	A control that allows you to display a visual element to the end user to show that a partial-page postback is occurring to the part of the page making the update. This is an ideal control to use when you have long-running AJAX updates.

12.4 THE UPDATEPANEL CONTROL

Microsoft's server-side AJAX framework consists of one main control: The UpdatePanel control. The UpdatePanel control is a container control, which means that it does not actually have UI-specific items associated with it. The UpdatePanel control enables you to update a portion of a page without updating the entire page. This control preserves the postback model and allows you to perform partial page rendering in a Web page, which in turn enhances the richness of the user interface, improving the application's performance and responsiveness. You can have multiple UpdatePanel controls in a Web page.

Let us start with a simple example of a page that uses the UpdatePanel control. The page contains a ScriptManager control and an UpdatePanel control. The UpdatePanel control contains a single Button control. When you click the button, only the content contained in the UpdatePanel control is refreshed.



Here is an example that illustrates how you can use the UpdatePanel control:

```
The <ContentTemplate> element
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server" />
Page Time: <%= DateTime.Now.ToString("T") %>
<br /><br />
<asp:UpdatePanel id="up1" runat="server">
<ContentTemplate>
UpdatePanel Time: <%= DateTime.Now.ToString("T") %>
<br />
<asp:Button id="btn" Text="Update" runat="server" />
</ContentTemplate>
</asp:UpdatePanel>
</form>
</body>
```

The code displays the current time both inside and outside the UpdatePanel control. When you click the button, only the time within the UpdatePanel control is refreshed.

Consider another example that includes AJAX. The following code does not use any of the ASP.NET AJAX controls. It contains two cascading DropDownList controls. The first DropDownList enables you to pick a state and the second DropDownList enables you to pick a city. The list of cities changes depending on the state selected.



```
<body>
<form id="form1" runat="server">
<div>
<asp:Label id="lblState" Text="State:" AssociatedControlID="ddlState" Runat="server" />
```

NOTES

NOTES

```

<asp:DropDownList id="ddlState" DataSourceID="srcState"
DataTextField="State"
DataValueField="State" AutoPostBack="true" Runat="server"
/>

<asp:SqlDataSource id="srcState" ConnectionString='<%$ConnectionStrings:con %>' SelectCommand="SELECT State FROM State ORDER BY State" Runat="server" />
<br /><br />
<asp:Label id="Label1" Text="City:" AssociatedControlID="ddlCity" Runat="server" />
<asp:DropDownList id="ddlCity" DataSourceID="srcCity" DataTextField="City" AutoPostBack="true" Runat="server" />
<asp:SqlDataSource id="srcCity" ConnectionString='<%$ConnectionStrings:con %>' SelectCommand="SELECT City FROM City WHERE State=@State ORDER BY City" Runat="server">
<SelectParameters>
<asp:ControlParameter Name="State" ControlID="ddlState" />
</SelectParameters>
</asp:SqlDataSource>
</div>
</form>
</body>

```

When you select a state using the first DropDownList control, there is a click, and the page posts back to it in order to populate the second DropDownList control with matching cities.

The UpdatePanel control has six important properties:

- (i) **ChildrenAsTriggers:** Gets or sets a Boolean value that indicates whether child controls should trigger an asynchronous postback automatically.
- (ii) **ContentTemplateContainer:** Gets the container for the UpdatePanel control's ContentTemplate. You can add controls to the ContentTemplate programmatically using this property.
- (iii) **IsInPartialRendering:** Gets a Boolean value indicating whether the UpdatePanel is being rendered in response to an asynchronous postback.
- (iv) **RenderMode:** Gets or sets a value that indicates whether the contents of an UpdatePanel should be enclosed in an HTML <div> or tag. Possible values are Block (the default) and Inline.
- (v) **Triggers:** Gets a list of controls that trigger the UpdatePanel to perform either an asynchronous or synchronous postback.

- (vi) **UpdateMode:** Gets or sets a value indicating when the content of the UpdatePanel is updated. Possible values are Always (the default) and Conditional.

The UpdatePanel also supports the following single important method:

- **Update () :** Causes the UpdatePanel to update its contents.

The UpdatePanel supports two types of triggers: AsyncPostBackTrigger and PostBackTrigger. The AsyncPostBackTrigger causes an asynchronous (AJAX) postback. The PostBackTrigger causes a normal entire-page postback.

NOTES

UpdatePanel Server-Side Page Execution Life Cycle

It is important to understand that a server-side page goes through its normal page execution life cycle when you perform an asynchronous postback. The Page PreInit, Init, Load and PreRender events are raised for an asynchronous postback in just the same way as these events are raised for a normal postback.

The server page life cycle includes events, such as:

1. PreInit
2. Init
3. Load
4. PreRender

UpdatePanel Client-Side Page Execution Life Cycle

A page that contains a ScriptManager control not only has a server-side page execution life cycle, it also has a client-side page execution lifecycle. The following series of events happen on the client-side (see Table 12.3):

Table 12.3 Events on the Client-Side

Event	Description
Application_init	It is raised when a page is first requested. This event is not raised during an asynchronous postback.
PageRequestManager.initializeRequest	It is raised before an asynchronous request to the server starts.
PageRequestManager.beginRequest	It is raised before an asynchronous request to the server starts.
PageRequestManager.pageLoading	It is raised after an asynchronous response is received from the server but before UpdatePanel content is updated.
PageRequestManager.pageLoaded	It is raised after an asynchronous response is received from the server and after UpdatePanel content is updated. Also it is raised during the initial page request.
Application_load	It is raised during both normal and asynchronous postbacks.
PageRequestManager.endRequest	It is raised after an asynchronous response both when there is and when there is not an error.
Application_unload	It is raised before the user leaves or reloads the page.

NOTES

12.5 THE Timer CONTROL

One common task when working with asynchronous postbacks from your ASP.NET pages is that you might want these asynchronous postbacks to occur at specific intervals in time. To accomplish this, you use the `Timer` control available to you from the AJAX Extensions part of the toolbox. The ASP.NET AJAX Timer control enables you to refresh an `UpdatePanel` (or the entire page) on a timed basis. The `Timer` control has one important property:

Interval: It is the amount of time, in milliseconds, between `Tick` events. The default value is 60,000 (1 minute).

The `Timer` control raises a `Tick` event every many milliseconds, depending on the value of its `Interval` property. If you do not associate the `Timer` control with an `UpdatePanel`, the `Timer` posts the entire page back to the server performing a normal postback. A simple example to demonstrate how this control works involves putting some timestamps on your page and setting postbacks to occur at specific timed intervals.

Timer.aspx Page

```
<%@ Page Language="C#" %>
<script runat="server">
protected void Page_Load(object sender, EventArgs e)
{
if (!Page.IsPostBack) {
Label1.Text = DateTime.Now.ToString();
}
}
protected void Timer1_Tick(object sender, EventArgs e)
{
Label1.Text = DateTime.Now.ToString();
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Timer Example</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

```

<ContentTemplate>
<asp:Label ID="Label1" runat="server" Text="Label"></
asp:Label>
<asp:Timer ID="Timer1" runat="server" OnTick="Timer1_Tick"
Interval="10000">
</asp:Timer>
</ContentTemplate>
</asp:UpdatePanel>
</div>
</form>
</body>
</html>

```

When this page loads for the first time, the `Label` control is populated with the `DateTime` value through the invocation of the `Page_Load` event handler. After this initial load of the `DateTime` value to the `Label` control, the `Timer` control takes care of changing this value.

The `OnTick` attribute from the `Timer` control enables you to accomplish this task. It points to the function that is triggered when the time span specified in the `Interval` attribute is reached.

The `Interval` attribute is set to 10000, which is, 10,000 milliseconds. This means, that every 10 seconds an asynchronous postback is performed and the `Timer1_Tick()` function is called. When you run this page, you will see the time change on the page every 10 seconds.

NOTES

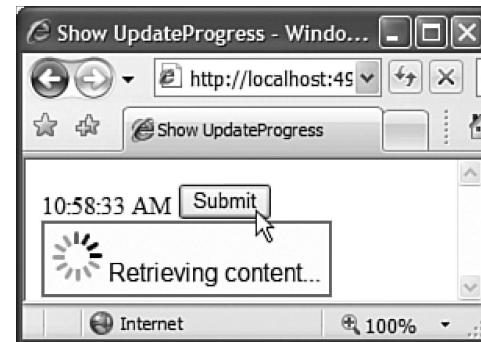
12.6 THE UpdateProgress CONTROL

The final server control that we need to examine in this chapter is the `UpdateProgress` control. This control enables you to display a progress indicator while an `UpdatePanel` is updating its content.

During a normal postback, the browser displays its progress in downloading new content by spinning an icon or displaying a progress bar. During an asynchronous postback, on the other hand, there is no visual indication of progress. You can use the `UpdateProgress` control to give the users some sense that something is happening during an asynchronous postback.

The following example illustrates how to use the `UpdateProgress` control. If you click the button, an animation spins while the asynchronous postback is performed.

NOTES



```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
protected void btnSubmit_Click(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(5000);
}
</script>
<html>
<head runat="server">
<title>Show UpdateProgress</title>
<style type="text/css">
.progress
{
    font-family:Arial;
    position: absolute;
    background-color:lightyellow;
    border:solid 2px red;
    padding:5px;
}
</style>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="up1" runat="server">
<ContentTemplate>
<%= DateTime.Now.ToString("T") %>
```

```

<asp:Button id="btnSubmit" Text="Submit" Runat="server"
    OnClick="btnSubmit_Click" />
</ContentTemplate>
</asp:UpdatePanel>
<asp:UpdateProgress ID="progress1"
    AssociatedUpdatePanelID="up1"
    runat="server">
    <ProgressTemplate>
        <div class="progress">
            <asp:Image id="imgProgress" ImageUrl="~/Images/
                Progress.gif" Runat="server" />
            Retrieving content...
        </div>
    </ProgressTemplate>
</asp:UpdateProgress>
</div>
</form>
</body>
</html>

```

Overview of AJAX

NOTES

When you click the button, the response is delayed for 5 seconds so you have a chance to see the progress indicator. The delay simulates a network delay.

The UpdateProgress control supports the following three properties:

- (i) **AssociatedUpdatePanelID:** The UpdateProgress control displays progress for this UpdatePanel control.
- (ii) **DisplayAfter:** It is the amount of time, in milliseconds, before the UpdateProgress control displays content. The default is 500 milliseconds (half a second).
- (iii) **DynamicLayout:** When this property is set to true (the default), the UpdateProgress control is initially hidden with the Cascading Stylesheet or CSS attribute `display:none`. When this property is set to false, the UpdateProgress control is hidden with the Cascading Stylesheet or CSS attribute `visibility: hidden`.

12.7 THE ASP.NET AJAX CONTROL TOOLKIT

The UpdatePanel, UpdateProgress and Timer controls are fairly useful. However, they are the only ASP.NET AJAX-enabled controls you will find in ASP.NET. Although there are only three controls that use ASP.NET AJAX features, ASP.NET actually includes a sophisticated library of JavaScript functions that can be used to create all sorts of advanced effects. One example is the ASP.NET AJAX Control Toolkit. The ASP.NET AJAX Control Toolkit is a joint

NOTES

project between Microsoft and the ASP.NET community. It consists of dozens of controls that use the ASP.NET AJAX libraries to create sophisticated effects.

The ASP.NET AJAX Control Toolkit consists of 34 server-side AJAX controls that you can use in your ASP.NET applications. You can take advantage of the controls to create Website special effects, such as animations, rounded corners and modal pop-ups.

The following are some of the merits of ASP.NET AJAX Control Toolkit:

- It is completely free.
- It includes full source code, which is helpful if you are ambitious enough to want to create your own custom controls that use ASP.NET AJAX features.
- It uses extenders that enhance the standard ASP.NET Web controls. That is why, you do not have to replace all the controls on your Web pages; instead, simply plug in the new bits of functionality that you need.

These advantages have generated a great deal of excitement around the ASP.NET AJAX Control Toolkit.

Downloading and Installing

The ASP.NET AJAX Control Toolkit is not included with the ASP.NET Framework. The Toolkit is being continuously updated every couple of months.

The Toolkit is maintained as a project at Microsoft CodePlex. You can download the latest release of the ASP.NET AJAX Control Toolkit at the following location:

<http://www.codeplex.com/AtlasControlToolkit>

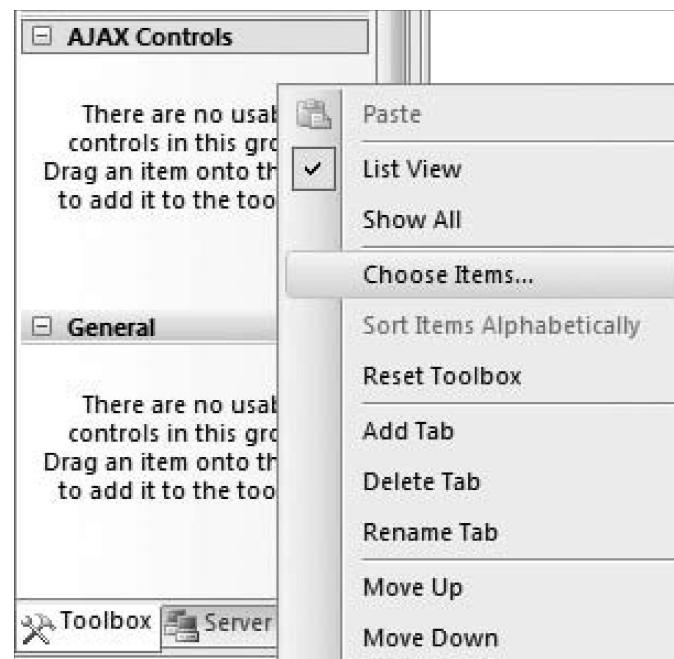
When you download the Toolkit, you can either download the controls and the source code or download the controls only. You will need to unzip the download onto your hard drive.

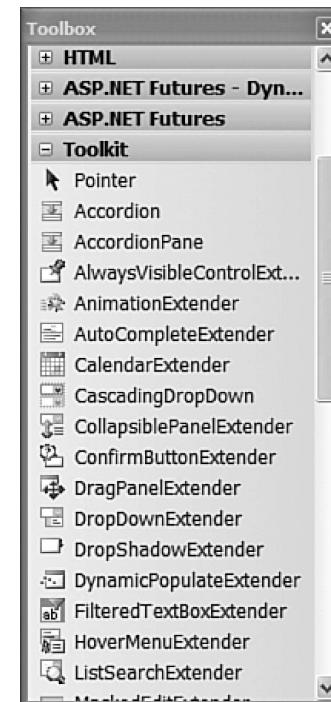
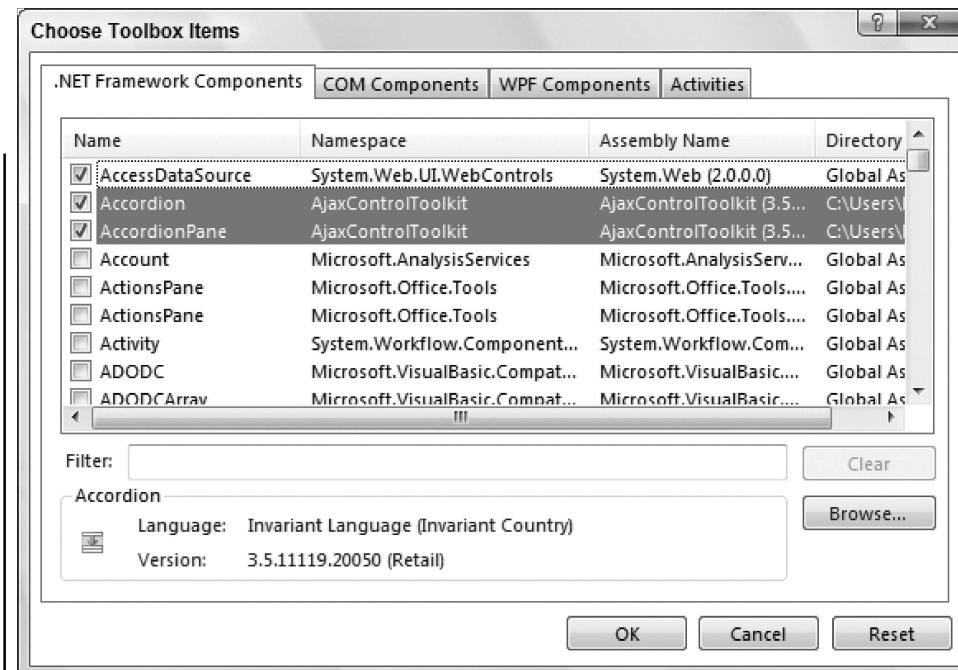
As part of the download, you get a sample Website that demonstrates each of the Toolkit controls. You can open the sample Website by launching Visual Web Developer, selecting the menu option File, Open Website and browsing to the SampleWebsite folder in the unzipped download.

The ASP.NET AJAX Control Toolkit is not installed in the Global Assembly Cache (GAC). You must copy the AjaxControlToolkit.dll assembly from the /Bin folder of the SampleWebsite to the /Bin folder in your application. There are multiple ways to do this:

- **Copy the Assembly by Hand:** You can simply copy the AjaxControlToolkit.dll assembly from the SampleWebsite /Bin folder to a /Bin folder located in a new Website.
- **Add an Assembly Reference:** Following are the steps:
 1. Within Visual Web Developer, select the menu option Website, Add Reference.

2. Select the Browse tab.
 3. Browse to the AjaxControlToolkit.dll assembly located in the SampleWebsite /Bin folder.
- **Add the Toolkit to Your Toolbox:** You can add the ASP.NET AJAX Control Toolkit to the Visual Web Developer Toolbox by following these steps:
1. Within Visual Web Developer, create a new ASP.NET page.
 2. Right-click in the Toolbox window and select the menu option Add Tab. Create a new tab named Toolkit.
 3. Right-click under the new tab and select the menu option Choose Items.
 4. Click the Browse button located at the bottom of the .NET Framework Components tab.
 5. Browse to the /Bin folder of the SampleWebsite and select the **AjaxControlToolkit.dll assembly.32**
 6. When you drag a control from the Toolbox onto a page, the **AjaxControlToolkit.dll** is copied to the Website /Bin folder automatically.

NOTES

NOTES

Now you can use the components from the ASP.NET AJAX Control Toolkit in any Web page in any Website. First, begin by adding the `ScriptManager` control to the Web page. Then, head to the new Toolbox tab you created and drag the ASP.NET AJAX control you want onto your page. The first time you do add a component from the ASP.NET AJAX Control Toolkit, Visual Studio will copy the `AjaxControlToolkit.dll` assembly to the Bin folder of your Web application, along with the localization assemblies.

Overview of the Toolkit Controls

Overview of AJAX

This section provides you a brief overview of each of the current Toolkit controls.

- **Accordion:** The Accordion control enables you to create a Microsoft Outlook-like expanding and collapsing menu. The Accordion control can contain one or more AccordionPane controls. One AccordionPane can be selected at a time. The selected pane is expanded and the other panes are collapsed.

NOTES

Accordion Demonstration

The screenshot shows a vertical Accordion control with three panes. The first pane, titled '1. Accordion', is expanded. The second pane, titled '2. AutoSize', is collapsed. The third pane, titled '3. Control or Extender', is collapsed. A tooltip for the 'Control or Extender' pane explains its functionality: 'The Accordion is written using an extender like most of the other extenders in the AJAX Control Toolkit. The extender expects its input in a very specific hierarchy of container elements (like divs), so the Accordion and AccordionPane web controls are used to generate the expected input for the extender. The extender can also be used on its own if you provide it appropriate input.'

4. What is ASP.NET AJAX?

Fade Transitions:

AutoSize: None

- **AlwaysVisibleControl:** The AlwaysVisibleControl enables you to display content that is fixed at one location in the browser window even when you scroll the window. The control works like the position:fixed Cascading Stylesheet or CSS attribute.
- **Animation:** The Animation control enables you to add fancy animation effects to your Website. For example, you can move, resize and fade elements in a page.
- **AutoComplete:** This control enables you to display suggestions as a user enters text into a text field.



- **Calendar:** The Calendar control displays a pop-up calendar next to a TextBox. It enables you to select a year, month and date by clicking dates in a pop-up calendar.

NOTES**Calendar with an associated button styled:**

- **CascadingDropDown:** The CascadingDropDown control enables you to make the list of items displayed in one DropDownList control dependent on the list of items displayed in another DropDownList control. The DropDownList items are updated by performing an asynchronous postback.
- **CollapsiblePanel:** This control enables you to hide or display content contained in a Panel control. When you click its header, the content either appears or disappears.
- **ConfirmButton:** This control enables you to display a confirmation dialog box when a user clicks a button. The confirmation dialog box can be the default JavaScript confirmation box.
- **DragPanel:** This control enables you to create a panel that you can drag with your mouse around the page. It enables you to create a virtual floating window.
- **DropDown:** The DropDown control enables you to create a SharePoint style dropdown menu.
- **DropShadow:** The DropShadow control enables you to add a drop shadow to a Panel control. You can set properties of the drop shadow, such as its width and opacity.
- **DynamicPopulate:** The DynamicPopulate control enables you to dynamically populate the contents of a control, such as a Label control, by performing an asynchronous call to the server.
- **FilteredTextBox:** The FilteredTextBox control enables you to prevent certain characters from being entered into a TextBox.
- **HoverMenu:** the HoverMenu displays a pop-up menu when you hover over another control.
- **ListSearch:** The ListSearch control enables you to perform an incremental search against the items in either a ListBox or DropDownList control.

- **MaskedEdit:** The MaskedEdit control forces a user to enter a certain pattern of characters into a TextBox control.
- **ModalPopup:** This control enables you to display a modal pop up. When the modal pop up appears, the remainder of the page is grayed out, preventing you from interacting with the page.
- **MutuallyExclusiveCheckBox:** The MutuallyExclusive CheckBox control enables you to treat a set of CheckBox controls like a set of RadioButton controls. Only one CheckBox can be selected at a time.
- **NoBot:** The NoBot control attempts to prevent spam robots from posting advertisements to your Website. The control attempts to detect whether a human or robot is posting content.
- **NumericUpDown:** The NumericUpDown control enables you to display up and down buttons next to a TextBox control. When you click the up and down buttons, you can cycle through a set of numbers or other items.
- **PagingBulletedList:** This control enables you to display different content depending on the bullet clicked in a BulletedList control.
- **PasswordStrength:** The PasswordStrength control enables you to display a pop up box that indicates the security strength of a password as a user selects a new password.
- **PopupControl:** The PopupControl displays a pop-up window.
- **Rating:** The Rating control enables you to rate an item on a scale.
- **ReorderList:** The ReorderList enables you to render an interactive list of items that supports reordering through drag and drop.
- **ResizableControl:** The ResizableControl enables you to resize images and other content contained on a Web page.
- **RoundedCorners:** The RoundedCorners control enables you to add rounded corners around an element on a page.
- **Slider:** The Slider control enables you to create either a horizontal or vertical slider for selecting a particular value in a range of values.
- **SlideShow:** The SlideShow control displays a slide show of images. The control can render Next, Previous, Play and Stop buttons.
- **Tabs:** The Tabs control enables you to create a tabbed interface. Switching tabs does not require a postback.
- **TextBoxWatermark:** The TextBoxWatermark control enables you to display background text inside of a TextBox control. When focus is given to the TextBox, the background text disappears.
- **ToggleButton:** The ToggleButton control enables you to customize the appearance of a CheckBox control.

NOTES

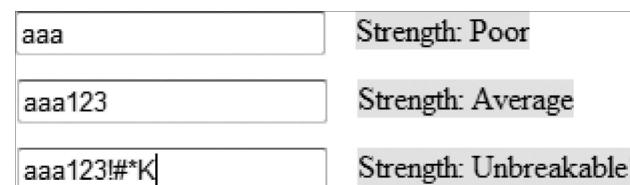
NOTES

- **UpdatePanelAnimation:** The UpdatePanelAnimation control enables you to display an animation while the UpdatePanel is performing an asynchronous postback.

- **ValidatorCallout:** The ValidatorCallout control can be used with any of the standard ASP.NET validation controls to create a callout effect when there is a validation error.

PasswordStrength Control

The PasswordStrength control allows you to check the contents of a password in a TextBox control and validate its strength. It will also then give a message to the end-user about whether the strength is reasonable.



```
<%@ Page Language="C#" %>
<%@ Register Assembly="AjaxControlToolkit"
Namespace="AjaxControlToolkit"
TagPrefix="cc1" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Password Strength Control</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<cc1:PasswordStrength ID="PasswordStrength1"
runat="server" TargetControlID="TextBox1">
</cc1:PasswordStrength>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</div>
</form>
</body>
</html>
```

Check Your Progress

1. What are the two major parts of AJAX framework?
2. What is server side framework?
3. Explain client side framework?

12.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. AJAX framework can be classified into two major parts:
 - (i) Server-side framework
 - (ii) Client-side framework
2. Server-side frameworks as the name suggests are used on the server-side. It is not actually JavaScript that is used in server-side framework, in fact you use the same language API (Application Programming Interface) is provided by the framework.
3. client-side framework is invoked within the user's browser. Most coding on AJAX today revolves around this type of framework.

NOTES

12.9 SUMMARY

- AJAX is a cross-platform combination of technologies that can be used to make your Web pages fast, rich and responsive.
- JavaScript that is used in server-side framework, in fact you use the same language Application Programming Interface is provided by the framework.
- The Update Panel control enables you to update a portion of a page without updating the entire page.
- The final server control that we need to examine in this chapter is the UpdateProgress control. This control enables you to display a progress indicator while an UpdatePanel is updating its content.
- The ASP.NET AJAX Control Toolkit consists of 34 server-side AJAX controls that you can use in your ASP.NET applications.
- One common task when working with asynchronous postbacks from your ASP.NET pages is that you might want these asynchronous postbacks to occur at specific intervals in time.
- A page that contains a ScriptManager control not only has a server-side page execution life cycle, it also has a client-side page execution lifecycle.
- Client-side framework is invoked within the user's browser. Most coding on AJAX today revolves around this type of framework. Probably, the best side of this framework is the ability to build widgets so that it could be integrated into a desktop environment.
- The key feature of AJAX can be seen when the client requests new information. Instead of sending a whole new page, AJAX sends only the required new information.

NOTES

-
- **AJAX:** It is a cross-platform combination of technologies that can be used to make Web pages faster, richer and responsive.
 - **ASP.NET AJAX Timer control:** It enables you to refresh an UpdatePanel (or the entire page) on a timed basis.
 - **The UpdateProgress control:** It enables you to display a progress indicator while an UpdatePanel is updating its content.
-

12.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Name the browsers that support the XMLHttpRequest object.
2. What are the properties of the XMLHttpRequest object? What are the different types of readyStates in AJAX?
3. How do you handle multiple or concurrent requests in AJAX?
4. What is the role of ScriptManager in AJAX?
5. Can we override the EnablePartialRendering property of the ScriptManager class?
6. How do you use multiple ScriptManager controls in a Web page?
7. What is an UpdatePanel Control?
8. What are the limitations of AJAX?
9. How do you ensure that the content of an UpdatePanel updates only when a partial postback takes place (and not on a full postback)?
10. How do you trigger a postback on an UpdatePanel from JavaScript?

Long Answer Questions

1. How do we create an XMLHttpRequest object for Internet Explorer? How is this different for other browsers?
2. What is the ASP.NET AJAX Framework? List the different versions.
3. What are the modes of updating in an UpdatePanel? What are Triggers of an UpdatePanel?

12.12 FURTHER READINGS

Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002.
Beginning VB.NET, 2nd edition. Indianapolis: Wiley Publishing Inc.

Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.

Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.

Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.

Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

NOTES

UNIT 13 INTRODUCTION TO DATABASE ACCESS

Structure

- 13.0 Introduction
 - 13.1 Objectives
 - 13.2 Database Access in the Internet World
 - 13.3 ADO.NET: An Overview
 - 13.4 The Connection Class
 - 13.5 The Command and DataReader Classes
 - 13.5.1 The Command Class
 - 13.5.2 The DataReader Class
 - 13.6 Answers to Check Your Progress Questions
 - 13.7 Summary
 - 13.8 Key Words
 - 13.9 Self Assessment Questions and Exercises
 - 13.10 Further Readings
-

13.0 INTRODUCTION

The programs written in any language, when executed, need some input data which are processed to produce output data. This input can be entered through keyboard each time the program runs, but entering data through keyboard is not suitable for programs requiring large amount of data. Thus, there must be some means through which we can store the data permanently so that the program can access that data. This is achieved through databases. Accessing a database in VB.NET requires database connectivity. Since, programmers are free to choose any database management system, such as MS Access, Oracle, SQL Server, etc., there must be some technology that allows database connectivity irrespective of the database system used. ADO.NET is one such technology of the .NET framework, which works with all types of data and databases. In this unit, you will learn about the accessing of databases with ADO.NET.

13.1 OBJECTIVES

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

- Manage data using database
- Create a database connection

- Explain the two types of objects in ADO.NET
- Understand the significance of `Connection` class

13.2 DATABASE ACCESS IN THE INTERNET WORLD

NOTES

The most common way to manage data is to use a database. Simply putting, database is a collection of homogeneous data in an organized manner. Database technology is particularly useful for business software, which typically requires sets of related information; for example, list of products and list of customers for sales. This type of information is best described using a relational model, which is the philosophy that underlies all modern database products, including SQL (Structured Query Language) Server, Oracle and even Microsoft Access.

As you probably know, a relational model breaks information down to its smallest and most concise units. For example, we have taken three tables, namely customer table, order table and order details table. The order details table does not store all the information about the orders. Instead, it stores just the order ID that refers to a full record in an order table and Customer ID that has full record of the customers.

Although it is possible to organize data into tables and store it in one or more files, this approach would not be very flexible. Instead, a Web application requires a full Relational Database Management System (RDBMS), such as SQL Server. The RDBMS handles the data infrastructure, disallowing invalid data, provides data to multiple users simultaneously, ensuring optimum performance and reliability. Figure 13.1 illustrate the RDBMS relationships.

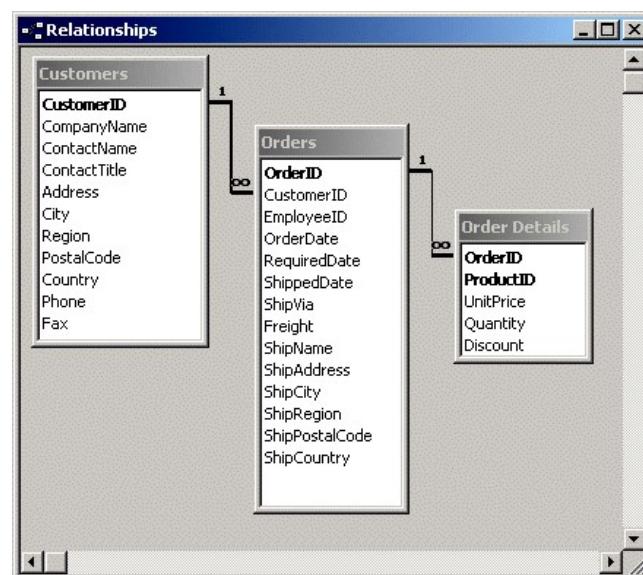


Fig.13.1 RDBMS Relationships

NOTES

In most ASP.NET applications, you will need to use a database for some tasks. Here are some basic examples of data at work in a Web application:

1. Search engines use databases to store page URLs, keywords and links.
2. E-commerce sites use databases to keep track of user information, orders, shipment records, and so on.
3. Media sites to store their articles and directories information in database.

Configuring Your Database

As described earlier, a database is a collection of homogeneous data in an organized manner. Before executing any data access code, you need a database server to run your command. Although there are many good options, all of which work equally well with ADO.NET, a majority of ASP.NET applications use Microsoft SQL Server.

If you do not have a full version of SQL Server, you can simply install the free SQL Server Express Edition. It includes all the database features you need to develop and test a Web application.

SQL Basics

When interacting with a data source through ADO.NET, you use SQL to retrieve, insert, modify and update information. However, for designing a database application efficiently, it is necessary to understand the basics of SQL.

SQL (Structured Query Language) is a database computer language designed for managing data in RDBMS, and originally based upon relational algebra. Simply we can say, it is a standard data access language used to interact with relational databases. Different databases differ in their support of SQL, but its scope includes data insert, select, add, and modify query and schema creation and modification.

When working with ADO.NET, however, you will probably use only the following standard types of SQL statements:

- A `SELECT` statement retrieves records.
- An `UPDATE` statement modifies existing records.
- An `INSERT` statement adds a new record.
- A `DELETE` statement deletes existing records.

Caching Database Data with `SqlDataSource` Control

Caching is useful primarily in situations where the data does not change frequently. The `SqlDataSource` control can cache data that it has retrieved, which can enhance the performance of your applications by avoiding the need to re-run resource dependent queries.

Additionally, when using the `SqlDataSource` control with the `System.Data.SqlClient` provider, you can make use of the

`SqlCacheDependency` object. This enables the `SqlDataSource` control to refresh the cache only if the data returned by the `SelectCommand` has been modified in the database.

Enabling Caching with the `SqlDataSource` Control

The `SqlDataSource` control can cache data when its `DataSourceMode` property is set to `DataSet`. Caching is not enabled by default, but you can enable it by setting the `EnableCaching` property to true.

Cached data is refreshed based on a time interval. You can set the `CacheDuration` property to the number of seconds to wait before the cache is refreshed. The `SqlDataSource` control maintains a separate cache item for each combination of `ConnectionString`, `SelectCommand`, and `SelectParameters` values. The following code example shows an `SqlDataSource` control configured to refresh data every 20 seconds:

```
<%@ Page language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional/
/EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html>
  <head runat="server">
    <title>Caching Example</title>
  </head>
  <body>
    <form id="form1" runat="server">

      <asp:SqlDataSource id="SqlDataSource1" runat="server"
        DataSourceMode="DataSet" ConnectionString="<%$ 
ConnectionStrings:MyNorthwind%"

        EnableCaching="True" CacheDuration="20"
        SelectCommand="SELECT EmployeeID, FirstName, LastName,
        Title FROM Employees">
      </asp:SqlDataSource>

      <asp:GridView id="GridView1" runat="server"
        AutoGenerateColumns="False"
        DataSourceID="SqlDataSource1">
        <columns>
          <asp:BoundField HeaderText="First
Name" DataField="FirstName" />
          <asp:BoundField HeaderText="Last
```

NOTES

NOTES

```
Name" DataField="LastName" />
<asp:BoundField HeaderText="Title" DataField="Title" />
</columns>
</asp:GridView>
</form>
</body>
</html>
```

13.3 ADO.NET: AN OVERVIEW

ADO.NET is a technology designed to let an ASP.NET program access data. It is a multi-layered architecture that contains Connection, Command and DataSet objects.

ADO.NET relies on the functionality in a small set of core classes. You can divide these classes into two groups:

- (i) Classes used to contain and manage data (such as DataSet, DataTable, DataRow and DataRelation)
- (ii) Classes used to connect to a specific data source (such as Connection, Command and DataReader)

The data container classes are completely generic. Whatever data source you use, once you extract the data, it is stored using the same data container: the DataSet class expertise. The DataSet is customized for relational data, which means it understands concepts, such as rows, columns and table relationships.

The second group of classes exists in several different flavors. Each set of classes of data interaction is called an ADO.NET data provider. ADO.NET data providers are personalized so that everyone can use the best-performing way of interacting with its data source. For example, the data provider of SQL Server is designed to work with SQL Server 7 or later version. Internally, it uses SQL Server's TDS (Tabular Data Stream) protocol for communication, thus, ensuring the best possible performance. If you are using Oracle, you will need to use the Oracle provider classes instead.

ADO.NET Data Providers

A data provider is a set of ADO.NET classes that enables you to access a database, execute SQL commands and retrieve data. Essentially, a data provider is a bridge between your application and a data source.

The following classes make up a data provider:

- **Connection:** You use this object to establish a connection to a data source.

NOTES

- **Command:** You use this object to execute SQL commands and stored procedures.
- **DataReader:** This object provides fast read only, forward-only access to the data retrieved from a query.
- **DataAdapter:** This object performs two tasks. First, you can use it to fill a DataSet (a disconnected collection of tables and relationships) with information extracted from a data source. Second, you can use it to apply changes to a data source.

The .NET Framework is bundled with a small set of four providers:

- **SQL Server Provider:** Provides optimized access to an SQL Server database (version 7.0 or later).
- **OLE DB Provider:** Provides access to any data source that has an OLE DB driver. This includes SQL Server databases prior to version 7.0.
- **Oracle Provider:** Provides optimized access to an Oracle database (version 8*i* or later).
- **ODBC Provider:** Provides access to any data source that has an ODBC driver.

Figure 13.2 shows ADO.NET data providers.

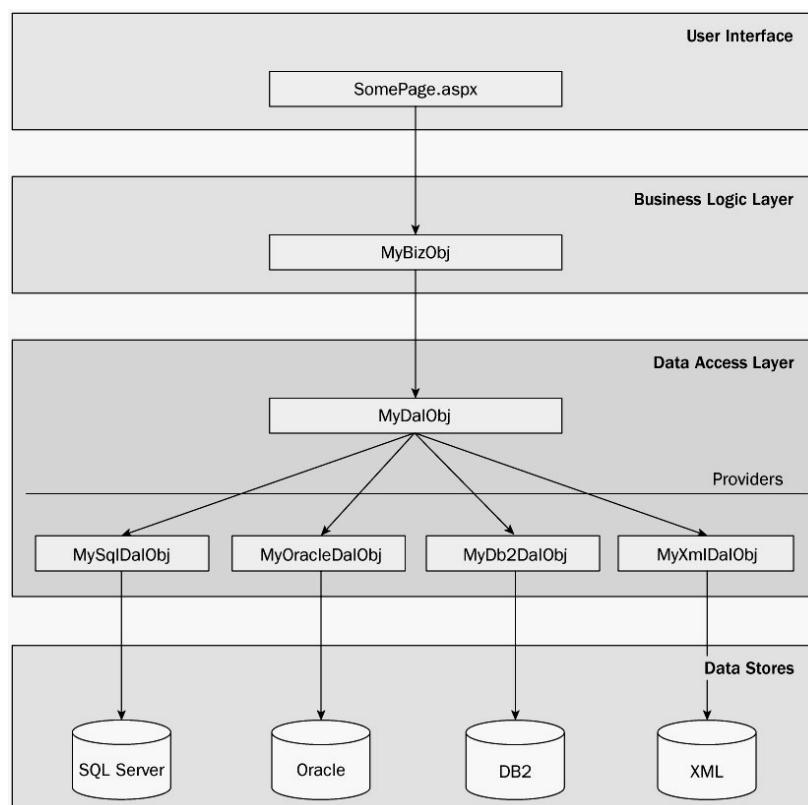


Fig. 13.2 ADO.NET Data Providers

NOTES

Data Namespaces

The following are the two types of objects in ADO.NET:

- **Connection Based Objects:** These are the data provider objects, such as Connection, Command, DataAdapter, and DataReader. They execute SQL statements, connect to a database or fill a DataSet. The connection based objects are specific to the type of data source.
- **Content Based Objects:** They include the DataSet, DataColumn, DataRow, DataRelation and several others. They are entirely independent of the type of data source and are found in the System.Data namespace.

The ADO.NET classes are grouped into several namespaces. Each provider has its own namespace and generic classes, such as the DataSet are stored in the System.Data namespaces. Together, these namespaces hold all the functionalities of ADO.NET (see Table 13.1).

Table 13.1 ADO.NET Namespaces

Namespace	Purpose
System.Data	Contains the key data container classes that model columns, relations, tables, datasets, rows, views and constraints. These classes are totally independent of any specific type of database.
System.Data.Common	Contains base abstract classes that define the core ADO.NET functionality. Not used directly in your code. These classes are used by other data provider classes that inherit from them.
System.Data.OleDb	Contains the classes you use to connect to an OLE DB data source and execute commands, including OleDbConnection and OleDbCommand.
System.Data.SqlClient	Contains the classes you use to connect to a SQL Server database and execute commands including SqlCommand, SqlConnection, and SqlDataAdapter.
System.Data.SqlTypes	Contains structures for SQL Server specific data types. These types are not required, but they provide an alternative standard .NET data types.
System.Data.OracleClient	Contains the classes you use to connect to an Oracle database and execute commands, such as OracleConnection and OracleCommand.
System.Data.Odbc	Contains the classes you use to connect to a data source through an ODBC driver and execute commands, such as OdbcConnection and OdbcCommand.

Before continuing, make sure you import the ADO.NET namespaces. Here, assume that you are using the SQL Server provider, in which case you need the following two namespace imports:

```
using System.Data;  
using System.Data.SqlClient;
```

Creating Database Connection

Step 1: Add Namespace “using System.Data.SqlClient;”

Step 2: Make SQL connection.

Write this code to create SQL connection.

```
SqlConnection con = new SqlConnection ("Server=Your server  
name;  
Database=Yourdatabasename;  
Trusted_Connection=True");  
SqlCommand cmd = new SqlCommand("select * from Table  
name");  
con.Open();  
DataSet ds = new DataSet(cmd,con);  
SqlDataAdapter da = new SqlDataAdapter();  
da.Fill(ds);  
con.Close();
```

NOTES

Adding ADO.NET Parameters

Each .NET Framework data provider included with the .NET Framework has a Command object. The .NET Framework Data Provider for OLE DB has the OleDbCommand object, the SQL Server data provider includes a SqlCommand object, and the Oracle data provider has an OracleCommand object. The first step in using parameters in SQL queries is to build a command string containing parameter placeholders. These placeholders are filled in with actual parameter values when the Command object executes.

ADO.NET Parameters in SQL Server

The syntax of an SQL Server parameter uses a “@” symbol prefix on the parameter name as shown below:

```
// Define the SQL Server command to get the product  
having the ID = 100  
SqlCommand command = new SqlCommand();  
command.CommandText = "SELECT * FROM Product WHERE  
Product.ID=@PROD_ID";  
command.Parameters.Add(new SqlParameter("@PROD_ID", 100));  
// Execute the SQL Server command  
SqlDataReader reader = command.ExecuteReader();  
DataTable tblProducts = new DataTable();  
tblProducts.Load(reader);  
foreach (DataRow rowProduct in tblProducts.Rows)  
{  
// Use the data...  
}
```

ADO.NET Parameters in Oracle

The syntax of an Oracle parameter uses a “:” symbol prefix on the parameter name as shown below:

```
// Define the Oracle command to get the product having  
the ID = 100
```

NOTES

```
OracleCommand command = new OracleCommand();
command.CommandText = "SELECT * FROM Product WHERE
Product.ID=:PROD_ID";
command.Parameters.Add(new OracleParameter (":PROD_ID",
100));
// Execute the Oracle command
OracleDataReader reader = command.ExecuteReader();
DataTable tblProducts = new DataTable();
tblProducts.Load(reader);
foreach (DataRow rowProduct in tblProducts.Rows)
{
    // Use the data...
}
```

13.4 THE CONNECTION CLASS

Before you can retrieve, insert or update data, you need to make a connection to the data source. The `Connection` class allows you to establish a connection to the data source that you want to interact with. The core `Connection` properties and methods are specified by the `IDbConnection` interface, which all `Connection` classes implement.

ConnectionString

When creating a `Connection` object, you need to specify a value for its `ConnectionString`. `ConnectionString` is a series of name/value settings separated by semicolons (;). It defines the basic information the computer needs to create a connection.

If you are using SQL Server 2005 Express Edition, your connection string will include an instance name.

Although connection strings vary based on the RDBMS and provider you are using, some amount of information regarding the following is almost always required:

- ***The Server where the Database is Located:*** In the examples in this book, it is considered that the database server is always located on the same computer as the ASP.NET application.
- ***The Database you Want to Use:*** Either you can create your own database or use the Northwind database, which is installed by default with most editions of SQL Server.
- ***How the Database should Authenticate You:*** The Oracle and SQL Server providers give you the choice of supplying authentication credentials.

The following is an example of a ConnectionString:

```
SqlConnection myConnection = new SqlConnection();  
myConnection.ConnectionString = @"DataSource=localhost\  
SQLEXPRESS;" +  
"Initial Catalog=Pubs;Integrated Security=SSPI";
```

The following list describes some of the most commonly used connection string properties, including the three properties used in the preceding example:

- **DataSource:** This indicates the name of the server where the data source is located. If the server is on the same computer that hosts the ASP.NET site, localhost is sufficient.
- **InitialCatalog:** This is the name of the database that this connection will be accessing.

It is only the “initial” database because you can change it later by using the `ConnectionString`. `ChangeDatabase()` method.

- **IntegratedSecurity:** This indicates that you want to connect to SQL Server using the Windows user account that runs the Web Page code.
- **ConnectionTimeout:** This determines how long your code will wait, in seconds, before generating an error if it cannot establish a database connection. The default `ConnectionTimeout` is of 15 seconds.

For example, here is the ConnectionString you would use to connect to the Northwind database on the current computer using Integrated Security (which uses the currently logged-in Windows user to access the database):

```
string connectionString = "DataSource=localhost;Initial  
Catalog=Northwind;" +  
"IntegratedSecurity=SSPI";
```

If Integrated Security is not supported, the connection must indicate a valid user and password combination. For a newly installed SQL Server database, the `sa` (system administrator) account is usually present. The following is a ConnectionString that uses this account:

```
string connectionString = "DataSource=localhost;Initial  
Catalog=Northwind;" +  
"user id=sa;password=opensesame";
```

Connection Pooling

A connection takes some amount of time to open. In a Web application in which requests are being handled frequently, connections will be opened and closed endlessly as new requests are processed. In this environment, opening and closing a connection can limit the performance of the system.

One solution is connection pooling.

NOTES

NOTES

Connection pooling is the method by which we keep a permanent set of open database connections to be shared by sessions using the same data source. This avoids the need to create and destroy connections all the time. Connection pools in ADO.NET are completely transparent to the programmer and your data access code need not be altered.

When a client requests a connection by calling `Open()` method, the connection is not re-created; rather, it is served directly from the available pool of connections. When a client releases a connection by calling `Close()` or `Dispose()`, it is not discarded but returned to the pool to serve the next request.

ADO.NET does not support a connection pooling mechanism. However, most vendors implement some form of ADO.NET connection pooling. The SQL Server and Oracle data providers implement their own efficient connection pooling algorithms. For a connection to be reused with SQL Server or Oracle, the connection string should match exactly. If it differs even slightly, a new connection will be created in a new pool.

With both the SQL Server and Oracle providers, connection pooling is enabled and used automatically. However, you can also use connection string parameters to configure pool size settings (see Table 13.2).

Table 13.2 Connection Pool Settings

Settings	Description
Max Pool Size	The maximum number of connections allowed in the pool (defaults to 100). If the maximum pool size has been reached, any further attempts to open a connection are queued until a connection becomes available.
Min Pool Size	The minimum number of connections always retained in the pool (defaults to 0). This number of connections will be created when the first connection is opened.
Pooling	When true (the default), the connection is drawn from the appropriate pool or, if necessary, is created and added to the appropriate pool.
Connection Lifetime	Specifies a time interval in seconds. If a connection is returned to the pool and its creation time is older than the specified lifetime, it will be destroyed. The default is 0, which disables this behavior. This feature is useful when you want to recycle a large number of connections at once.

Some vendors provide methods for emptying out the connection pool. With the `SqlConnection` you can call the static `ClearPool()` and `ClearAllPools()` methods. When you call the `ClearPool()` method, you supply a `SqlConnection`, and all the matching connections are removed. `ClearAllPools()` removes every connection pool in the current application domain. However, these methods do not close the connections. They just mark them as invalid. This functionality is rarely used.

Connection Statistics

If you have the SQL Server provider connection, you can get some interesting statistics using the `SqlConnection.RetrieveStatistics()` method. `RetrieveStatistics` returns a hash table along with various details that can help you analyse the performance of commands and the amount of work you have done. Connection statistics are rarely used in a deployed application, but

they are useful for diagnosing performance during the testing and profiling phase. By default, connection statistics are disabled to improve performance. To use connection statistics, you need to set the `SqlConnection.StatisticsEnabled` property to true.

The following is an example that displays the number of bytes received by the connection since you enabled statistics:

```
Hashtable statistics = con.RetrieveStatistics();
lblBytes.Text = "Retrieved bytes: " + statistics
["BytesRetrieved"].ToString();
```

Statistics are provided in a loosely typed name/value collection. In order to retrieve it, you need to know the specific name of a statistic. You can find the full list in the MSDN help, but here are a few of the most useful:

- **ServerRoundtrips**: Indicates the number of times the connection has made a request to the database server. Typically, this value corresponds to the number of commands you have executed.
- **ConnectionTime**: Indicates the cumulative amount of time the connection has been open.
- **BytesReceived**: Indicates the total number of bytes retrieved from the database server.
- **SumResultSets**: Indicates the number of queries you have performed.
- **SelectRows**: Records the total number of rows retrieved in every query you have executed.

NOTES

13.5 THE Command AND DataReader CLASSES

In this, we will discuss the command and DataReader classes in detail.

13.5.1 The Command Class

The Command class helps you to execute any type of SQL statement. With the Command class, you are much more likely to perform data-manipulation tasks (such as retrieving and updating the records in a table), rather than performing data-definition tasks (such as creating and altering databases, tables and indexes).

The provider-specific Command classes implement standard functionality, just like the Connection classes. In this case, the `IDbCommand` interface defines the core set of Command methods that are used to execute a command over an open connection.

Before you can give a command, you need to select the command type, set the command text and bind the command to a connection. You can perform this by setting the corresponding properties or you can pass the information as a constructor argument.

The command text can be an SQL statement, a stored procedure or the name of a table. It all depends on the type of command you use. Three types of commands exist. These are shown in Table 13.3.

NOTES

Table 13.3 Types of Command Class

Value	Description
CommandType.Text	The SQL statement is provided in the CommandText property. This is the default value.
CommandType.StoredProcedure	The command will execute a stored procedure in the data source. The CommandText property provides the name of the stored procedure.
CommandType.TableDirect	The command will query all the records in the table. The CommandText is the name of the table from which the command will retrieve the records.

For example, here is how you would create a Command object that represents a query:

```
SqlCommand cmd = new SqlCommand();  
cmd.Connection = con;  
cmd.CommandType = CommandType.Text;  
cmd.CommandText = "SELECT * FROM Employees";
```

The following is a more efficient way using one of the Command constructors. Note that you do not need to specify the CommandType, because CommandType.Text is the default.

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees",  
con);
```

Alternatively, to use a stored procedure, you would use a code, such as follows:

```
SqlCommand cmd = new SqlCommand("GetEmployees", con);  
cmd.CommandType = CommandType.StoredProcedure;
```

Table 13.4 Command Methods

Method	Description
ExecuteNonQuery()	Executes non-select commands, such as SQL commands that insert, delete, or update records. The returned value indicates the number of rows affected by the command.
ExecuteScalar()	Executes a SELECT query and returns the value of the first field of the first row from the rowset generated by the command.
ExecuteReader()	Executes a SELECT query and returns a DataReader object that wraps a read-only, forward-only cursor.

13.5.2 The DataReader Class

A DataReader allows you to retrieve one record at a time returned by a SELECT statement. This is called a firehose cursor. Using a DataReader is the simplest way to get to your data, but it lacks the ability to sort disconnected data. However, the DataReader provides the quickest possible, no-nonsense access to data. Table 13.5 shows the DataReader methods.

Table 13.5 DataReader Methods

Methods	Description
Read()	Advances the row cursor to the next row in the stream. This method must also be called before reading the first row of data. The Read() method returns true if there's another row to be read, or false if it's on the last row.
GetValue()	Returns the value stored in the field with the specified column name or index, within the currently selected row.
GetValues()	Saves the values of the current row into an array. The number of fields that are saved depends on the size of the array you pass to this method.
GetInt32(), GetChar(), GetDateTime()	These methods return the value of the field with the specified index in the current row, with the data type specified in the method name.
NextResult()	If the command that generated the DataReader returned more than one rowset, this method moves the pointer to the next rowset just before the first row.
Close()	Closes the reader.

NOTES**The ExecuteReader() Method**

The ExecuteReader() method sends the CommandText to the Connection and builds an SqlDataReader.

The following example creates a simple query command to return all the records from the Employees table in the sample database. The command is created on the button click.

```
protected void Button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection("server = MACT114;
    uid=sa;pwd=123; database=Sample;");
    SqlCommand cmd = new SqlCommand();

    cmd.CommandText = "select * from Employees";
    cmd.Connection = con;
    con.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    Response.Write("<table border=1>");
    while (dr.Read())
    {
        Response.Write("<tr>");
        Response.Write("<td>" + dr[0].ToString() + "</td><td>" +
        dr[1].ToString() + "</td><td>" + dr[2].ToString() + "</td><td>" +
        dr[3].ToString() + "</td>");
        Response.Write("</tr>");
    }

    dr.Close();
    con.Close();
}
```

Once you have the DataReader, you can loop through its records by calling the `Read()` method in a while loop. This moves the row cursor to the next record. The `Read()` method also returns a Boolean value indicating whether there are more rows to read.

CommandBehavior

The `ExecuteReader()` method takes one of the values from the `CommandBehavior` as a parameter. One useful value is `CommandBehavior.Close Connection`. When you pass this value to the `ExecuteReader()` method, the DataReader closes the associated connection as soon as the DataReader is closed.

Using this technique, you could rewrite the code as follows:

```
SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.  
CloseConnection);  
// (Build the HTML string here.)  
// No need to close the connection. You can simply close  
// the reader.  
reader.Close();  
HtmlContent.Text = htmlStr.ToString();
```

This behavior is of use when you retrieve a DataReader in one method and pass it to another method to process it. If you use the `CommandBehavior.CloseConnection` value, the connection will be automatically closed as soon as the second method closes the reader.

Processing Multiple Result Sets

A command can return more than one result set in two ways:

- (i) If you are calling a stored procedure, it may use multiple `SELECT` statements.
- (ii) If you are using a straight text command, you may be able to batch multiple commands by separating commands with a semicolon (;). Not all providers support this technique, but the SQL Server database provider does.

The following is an example of a string that defines a batch of three `SELECT` statements:

```
string sql = "SELECT TOP 5 * FROM Employees;" +  
"SELECT TOP 5 * FROM Products; SELECT TOP 5 * FROM Orders";
```

This string contains three queries. Together, they return the first five records from the `Employees` table, the first five from the `Products` table and the first five from the `Orders` table. First, the DataReader will provide access to the results from the `Employees` table. Once you have finished using the `Read()` method to read all these records, you can call `NextResult()` to view the next result set. When there are no more result sets, this method returns false.

The **ExecuteScalar()** Method

The `ExecuteScalar()` method returns the value stored in the first field of the first row of a result set generated by the command's `SELECT` query. This method is usually used to execute a query that retrieves only a single field; for example, SQL functions, such as `COUNT()` or `SUM()`.

The following procedure selects the maximum basic salary of an employee from the `salary_details` table.

```
protected void Button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection("server = MACT114;
uid=sa;pwd=123; database=Sample;");
    SqlCommand cmd = new SqlCommand("select MAX(ebasic) from
salary_details", con);
    con.Open();
    int result = Convert.ToInt32(cmd.ExecuteScalar());
    Response.Write(result);
    con.Close();
}
```

The **ExecuteNonQuery()** Method

The `ExecuteNonQuery()` method executes commands that do not return a result set, such as `INSERT`, `DELETE` and `UPDATE`. The `ExecuteNonQuery()` method returns a single piece of information.

This example updates the record of the table `Categories` using `ExecuteNonQuery`.

```
protected void Button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection("server = MACT114;
uid=sa;pwd=123; database=Sample;");
    SqlCommand cmd = new SqlCommand(" insert into Categories
values ('Machines')",con);

    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    Response.Write("the table has been altered.....!!!!");
}
```

NOTES

Check Your Progress

1. What is the purpose of designing the ADO.NET?
2. What are the two types of objects in ADO.NET?
3. What is the use of Connection class?

13.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. ADO.NET is a technology designed to let an ASP.NET program access data.
2. Connection based objects and content based objects are the two types of objects in ADO.NET.
3. The Connection class allows you to establish a connection to the data source that you want to interact with.

13.7 SUMMARY

- The most common way to manage data is to use a database. Simply putting, database is a collection of homogeneous data in an organized manner.
- When interacting with a data source through ADO.NET, you use SQL to retrieve, insert, modify and update information.
- ADO.NET is a technology designed to let an ASP.NET program access data. It is a multi-layered architecture that contains Connection, Command and DataSet objects.
- A data provider is a set of ADO.NET classes that enables you to access a database, execute SQL commands and retrieve data.
- Connection Based Objects are the data provider objects, such as Connection, Command, DataAdapter, and DataReader. They execute SQL statements, connect to a database or fill a DataSet.
- Content Based Objects includes the DataSet, DataColumn, DataRow, DataRelation and several others. They are entirely independent of the type of data source and are found in the System.Data namespace.
- The Connection class allows you to establish a connection to the data source that you want to interact with. The core Connection properties and methods are specified by the IDbConnection interface, which all Connection classes implement.

- The `Command` class helps you to execute any type of SQL statement.
- A `DataReader` allows you to retrieve one record at a time returned by a `SELECT` statement. This is called a firehose cursor.

NOTES

13.8 KEY WORDS

- **Database:** It is a collection of homogeneous data in an organized manner.
- **SQL:** It is a database computer language designed for managing data in relational database management systems (RDBMS).
- **ConnectionString:** It is a series of name/value settings separated by semicolons (;) and defines the basic information that the computer needs to create a connection.

13.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What is the function of `COMMIT`?
2. When does a transaction end?
3. What is a read-only transaction?
4. Is it possible to read from a `SqlDataReader` instance that is closed?
5. Name the properties which can be called after the `SqlDataReader` is closed.
6. What is the default position and default value of `SqlDataReader`?

Long Answer Questions

1. Explain the process of caching database data with `SQLDataSource` control.
2. Explain the various classes of ADO.NET.
3. Explain the significance of ADO.NET data providers.
4. Is it possible to execute multiple queries by using `DataReader`? Explain in detail.

13.10 FURTHER READINGS

Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002.
Beginning VB.NET, 2nd edition. Indianapolis: Wiley Publishing Inc.

NOTES

- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.

UNIT 14 SQL BASICS

Structure

- 14.0 Introduction
- 14.1 Objectives
- 14.2 Data Components and the Dataset
- 14.3 Disconnected Data
- 14.4 The DataSet Class
- 14.5 The DataTable Class
- 14.6 The DataRow Class
- 14.7 DataColumn Class
- 14.8 Answers to Check Your Progress Questions
- 14.9 Summary
- 14.10 Key Words
- 14.11 Self Assessment Questions and Exercises
- 14.12 Further Readings

NOTES

14.0 INTRODUCTION

In professional applications, your data access code is never embedded directly in the code-behind for a page. Instead, it is separated into a dedicated data component. In this chapter, you will see how to create a simple data access class of your own, adding a separate method for each data task you need to perform.

This chapter also tackles disconnected data—the ADO.NET features that revolve around the DataSet and allow you to interact with data even if you have closed the connection to the data source.

ASP.NET introduced a rich set of **data bound server controls**. Data bound controls allow you to define a declarative link between your page and a data source (such as a database). Data source controls are notable for the way they plug into the data binding infrastructure. In fact, by using a data source control, you can create a sophisticated page that allows you to query and update a database—all without writing a single line of ADO.NET code.

14.1 OBJECTIVES

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

- Create and design a data class
- List the benefits of stored procedures
- Create a DataSet object class
- Explain the significance of data binding
- Give an overview of data bound controls
- Display data with GridView control

14.2 DATA COMPONENTS AND THE DATASET

The topic ‘**Data Binding Control**’ is already discussed in Unit 11.

NOTES

Creating a Data Class

In duly organized applications, database code is not integrated directly in the client but hidden in a dedicated class. To perform a database operation, the client creates an instance of this class and calls the appropriate method. Follow these basic guidelines to create a Data class:

- **Connections should be opened and closed quickly:** Open the database connection in each and every method call, and close the connection before the method ends. The connections should not be open between client requests and the client should not have any control over how connections are acquired or when they are released. If the client introduces the possibility that a connection cannot be closed as soon as possible, it hampers scalability.
- **Implement error handling:** Use error handling to make sure the connection is closed even if the SQL (Structured Query Language) command generates an exception.
- **Stateless design approach:** The information necessary for a method is accepted in its parameters and the retrieved data is returned through the return value.
- **Preventing client from specifying connection string information:** This poses security risks and compromises the ability of connection pooling, which requires matching connection strings.
- **Do not connect with the client’s User ID:** Introducing any variability into the connection string will foreclose connection pooling.
- **Do not let the client use wide-open queries:** Every query should prudently select only the columns it needs.

Good design, easy for a database component uses a separate class for each database table. The database methods commonly accessible, such as insert, delete and modify, are wrapped separately in the stateless methods. Finally, all database calls use a dedicated stored procedure.

Layered Database Design

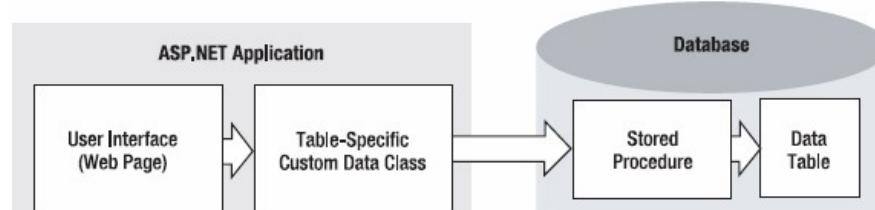


Fig. 14.1 Layered Database Design

Figure 14.1 demonstrates a simple database component. Instead of putting the database code in the Web page, component separates the code into a distinct class that can be used in multiple pages. The data component actually consists of two classes—a data package class that wraps a single record of information and a database utility class that performs the actual database operations with ADO.NET code.

NOTES

Data Package

The following is an example of EmployeeDetails class that provides all the fields as public properties. This class does not include all the information that is in the Employees table in order to make the example more concise.

```
public class EmployeeDetails
{
    private int employeeID;
    public int EmployeeID
    {
        get { return employeeID; }
        set { employeeID = value; }
    }
    private string firstName;
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    private string lastName;
    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
    public EmployeeDetails(int employeeID, string
        firstName, string lastName)
    {
        this.employeeID = employeeID;
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

NOTES**Stored Procedure**

Before you can start coding the data access logic, you need to make sure you have the set of stored procedures you need in order to retrieve, insert and update information. The following code shows the five stored procedures that are needed:

```
CREATE PROCEDURE InsertEmployee
@EmployeeID int OUTPUT
@FirstName varchar(10),
@LastName varchar(20),
AS
INSERT INTO Employees (LastName, FirstName,
HireDate) VALUES (@LastName, @FirstName,
GETDATE());
SET @EmployeeID = @@IDENTITY
GO
CREATE PROCEDURE DeleteEmployee
@EmployeeID int
AS
DELETE FROM Employees WHERE EmployeeID =
@EmployeeID
GO
CREATE PROCEDURE UpdateEmployee
@EmployeeID int,
@LastName varchar(20),
@FirstName varchar(10)
AS
UPDATE Employees SET LastName = @LastName,
FirstName = @FirstName
WHERE EmployeeID = @EmployeeID
GO
CREATE PROCEDURE CountEmployees
AS
SELECT COUNT(EmployeeID) FROM Employees
GO
CREATE PROCEDURE GetEmployee
@EmployeeID int
AS
SELECT FirstName, LastName FROM Employees
WHERE EmployeeID = @EmployeeID
GO
```

Finally, you need the utility class that performs the real database operations. This class uses the stored procedures that were shown in the previous section.

In this example, the data utility class is named EmployeeDU. It encapsulates all the data access codes and specific details of database. Here is the basic outline:

```
public class EmployeeDU
{
    private string connectionString;
    public EmployeeDU()
    {
        // Get default connection string.
        connectionString =
            WebConfigurationManager.ConnectionStrings[
                "Northwind"].ConnectionString;
    }
    public EmployeeDU(string
        connectionStringName)
    {
        // Get the specified connection string.
        connectionString =
            WebConfigurationManager.ConnectionStrings[
                "connectionStringName"].ConnectionString;
    }
    public int InsertEmployee(EmployeeDetails
        emp)
    { ... }
    public void DeleteEmployee(int employeeID)
    { ... }
    public void UpdateEmployee(EmployeeDetails emp)
    { ... }
    public EmployeeDetails GetEmployee()
    { ... }
    public int CountEmployees()
    { ... }
}
```

NOTES

You may have observed that the EmployeeDU class uses instance methods, not static methods. This is because although the EmployeeDU class does not store any state from the database; it does store the connection string as a private member variable. Because this is an instance class, the connection string can be retrieved every time the class is created, rather than every time a method is

invoked. This approach makes the code precise and allows it to be slightly faster (by avoiding the need to read the web.config file multiple times).

The code for inserting a record is as follows:

NOTES

```
public int InsertEmployee(EmployeeDetails emp)
{
    SqlConnection con = new
        SqlConnection(connectionString);
    SqlCommand cmd = new
        SqlCommand("InsertEmployee", con);
    cmd.CommandType =
        CommandType.StoredProcedure;
    cmd.Parameters.Add(new
        SqlParameter("@FirstName",
        SqlDbType.NVarChar, 10));
    cmd.Parameters["@FirstName"].Value =
        emp.FirstName;
    cmd.Parameters.Add(new
        SqlParameter("@LastName",
        SqlDbType.NVarChar, 20));
    cmd.Parameters["@LastName"].Value =
        emp.LastName;
    cmd.Parameters.Add(new
        SqlParameter("@EmployeeID", SqlDbType.Int,
        4));
    cmd.Parameters["@EmployeeID"].Direction =
        ParameterDirection.Output;
    try
    {
        con.Open();
        cmd.ExecuteNonQuery();
        return (int)cmd.Parameters["@EmployeeID"].Value;
    }
    catch (SqlException err)
    {
        // Replace the error with something less
        // specific.
        // You could also log the error now.
        throw new ApplicationException("Data error.");
    }
    finally
    {
        con.Close();
    }
}
```

As you can see, the method accepts the data using the package `EmployeeDetails`. Errors are caught and sensitive internal details are not returned to Web page code. This prevents the Web page from providing information that could lead to potential vulnerabilities.

NOTES

14.3 DISCONNECTED DATA

The examples you covered so far used ADO.NET connection-based features. ADO.NET focuses on an entirely different notion of the `DataSet` object. Disconnected data can be used to communicate between distributed applications or Web services. Like connected data, disconnected data supports multiple tables and data binding. In addition, it also supports forward and backward navigation in a result set.

With disconnected data access, you keep a copy of your data in memory using the `DataSet`. Once the database connection is opened, fetch your data and dump it into the `DataSet`, and disconnect immediately.

If you alter the information in the `DataSet`, the information in the corresponding table in the database does not change. This implies that you can easily process and manipulate the data without worry, because you are not using a valuable database connection.

Some of the scenarios in which a `DataSet` is easier to use than a `DataReader` include the following:

- When you want a favorable package to send the data to another component.
- When you need a convenient file format to serialize the data to disk.
- When you want to navigate backward and forward through a large amount of data.
- When you want to navigate from one table to another. Using the `DataSet`, you can store several tables of information. You can even define relationships that allow you to browse through them more efficiently.
- Navigation is not possible with the `DataReader`, which goes in one direction only—forward.
- When you want to manipulate the data as XML (Extensible Markup Language).

Storing `ConnectionString` in Web Configuration File

It is a good practice to store the connection string for your application in a config file rather than as a hard coded string in your code. Do not use appsettings in **web.config**. Instead, use the `ConnectionStrings` section in **web.config**.

```
<ConnectionStrings>
<add name="myConnectionString">
```

NOTES

```
ConnectionString="server=localhost; database=myDb;
uid=myUser; password=myPass;" />
</connectionStrings>
```

To read the connection string into your code, use the ConfigurationSettings class.

```
string connstr =
ConfigurationManager.ConnectionStrings["myConnectionString"].
ConnectionString;
```

Executing Stored Procedures

A stored procedure is an already written SQL statement that is saved in the database. Microsoft SQL Server provides the stored procedure mechanism to simplify the database development process by grouping Transact-SQL statements into manageable blocks.

Benefits of Stored Procedures

The following are the key benefits of stored procedures:

- **Precompiled Execution:** SQL Server compiles each stored procedure once and then reutilizes the execution plan. This results in tremendous performance boosts when stored procedures are called repeatedly.
- **Reduced Client/Server Traffic:** If network bandwidth is a concern in your environment, stored procedures can reduce long SQL queries to a single line that is transmitted over the wire.
- **Efficient Reuse of Code and Programming Abstraction:** Stored procedures can be used by multiple users and client programs.
- **Enhanced Security Controls:** You can grant users permission to execute a stored procedure independently of underlying table permissions.

Example of a stored procedure

Let us create a procedure called `sp_AddNewEmployee`. The SQL code is as follows:

```
CREATE proc sp_AddNewEmployee
(
    @EmpName varchar(32),
    @EmpCompany varchar(32),
    @EmpAddress varchar(100),
    @EmpEmailid varchar(100)
)
as
begin
    insert into
        d_EmployeeMaster(Empname, Empcompany,
```

```
EmpAddress, EmpEmailId)
values (@EmpName, @EmpCompany, @EmpAddress, @EmpEmailid)
select * from d_EmployeeMaster order by empid
desc
end
```

SQL Basics

NOTES

Using the ASP.NET **CookieParameter** Object

You can use the **CookieParameter** class to bind the value of a client-side HTTP (Hyper Text Transfer Protocol) cookie passed as part of an HTTP request to a parameter used by ASP.NET data source controls.

The **CookieParameter** class provides the `CookieName` property which identifies the name of the `HttpCookie` object to bind to, in addition to those inherited from the `Parameter` class. The **CookieParameter** class attempts to bind to the named cookie every time the `Evaluate` method is called.

The following example demonstrates how to use a `SqlDataSource` control and **CookieParameter** object bound to an HTTP cookie to display data from the Northwind Traders database in a `GridView` control:

```
<script language = "C#" runat = "server">
    void Page_Load(Object src, EventArgs e)
    {
        if ( !IsPostBack ) {
            Response.Cookies.Add ( new HttpCookie
( "lname", "davolio" ) );
            Response.Cookies.Add ( new HttpCookie
( "loginname", "ndavolio" ) );
            Response.Cookies.Add ( new HttpCookie (
"lastvisit", DateTime.Now.ToString ( ) ) );
        }
    }
</script>
<asp:sql datasource id = "sql" runat = "server"
SelectCommand = "SELECT OrderID, OrderDate,
CustomerID, ShipCity, ShipCountry FROM nw_Orders
WHERE EmployeeID = (SELECT EmployeeID FROM
nw_Employees WHERE LastName = @lastname)"
connectionstring = "<%$ ConnectionStrings:asp
%>">
    <selectparameters>
        <asp:cookieparameter name = "lastname"
cookiename = "lname" />
    </selectparameters>
</asp:sql datasource>
<asp:gridview id = "ordersGrid" runat = "server"
```

```
datasourceid = "sql" AllowPaging="true">
</asp:gridview>
```

NOTES**14.4 THE DataSet CLASS**

Because ADO.NET is all about scalability and performance, the disconnected mode is the preferred way to code client/server applications. Instead of a simple disconnected record set, ADO.NET gives you the `DataSet` object, the heart of disconnected data access, much like a small relational database held in memory on the client. It implements a database that only resides in the memory and is loose from any other data source. It provides you the ability to create multiple tables, fill them with data coming from different sources, enforce relationships between pairs of tables, and more.

A `DataSet` is an in-memory data store that can hold numerous tables. It contains two important ingredients: a collection of zero or more tables and a collection of zero or more relationships that you can use to link tables together. The tables in a `DataSet` are implemented by the `DataTable` class.

`DataSet` only hold data and do not interact with a data source. It is the `SqlDataAdapter` that manages connections with the data source and gives us disconnected behavior. The `SqlDataAdapter` opens a connection only when required and closes it as soon as it has performed its task. The `SqlDataAdapter` performs the following tasks when filling a `DataSet` with data:

1. Open connection.
2. Retrieve data into `DataSet`.
3. Close connection.

Moreover, it performs the following actions when updating data source with `DataSet`:

1. Open connection.
2. Write changes from `DataSet` to `DataSource`.
3. Close connection.

Creating a `DataSet` object

```
DataSet dsCustomers = new DataSet();
```

The `DataSet` constructor does not require parameters.

Creating a `SqlDataAdapter`

The `SqlDataAdapter` holds the SQL commands and connection object for reading and writing data. You initialize it with an SQL select statement and connection object:

```
SqlDataAdapter daCustomers = new  
SqlDataAdapter(  
    "select CustomerID, CompanyName from  
    Customers", conn);
```

SQL Basics

Filling the **DataSet**

NOTES

Once you have a **DataSet** and **SqlDataAdapter** instances, you need to fill the **DataSet**. You can fill the **DataSet** by using the **Fill** method of the **SqlDataAdapter**:

```
daCustomers.Fill(dsCustomers, "Customers");
```

The **Fill** method, in the code above, takes two parameters: a **DataSet** and a table name. The **DataSet** must be instantiated before trying to fill it with data. The second parameter is the name of the table that will be created in the **DataSet**.

Using the **DataSet**

A **DataSet** will bind with both **ASP.NET** and **Windows forms DataGrids**. An example that assigns the **DataSet** to a **Windows forms DataGrid** is as follows:

```
dgCustomers.DataSource = dsCustomers;  
dgCustomers.DataMember = "Customers";
```

Implementing a Disconnected Data Management Strategy

```
using System;  
using System.Data;  
using System.Data.SqlClient;  
using System.Windows.Forms;  
class DisconnectedDataform : Form  
{  
    private SqlConnection conn;  
    private SqlDataAdapter daCustomers;  
    private DataSet dsCustomers;  
    private DataGrid dgCustomers;  
    private const string tableName = "Customers";  
    // initialize form with DataGrid and Button  
    public DisconnectedDataform()  
    {  
        // fill dataset  
        Initdata();  
        // set up datagrid  
        dgCustomers = new DataGrid();
```

NOTES

```
dgCustomers.DataSource = dsCustomers;
dgCustomersDataMember = tableName;
// create update button
Button btnUpdate = new Button();
btnUpdate.Text = "Update";

// make sure controls appear on form
Controls.AddRange(new Control[] {
dgCustomers, btnUpdate });
}

// set up ADO.NET objects
public void Initdata()
{
    // instantiate the connection
    conn = new SqlConnection(
        "Server=(local); DataBase=Northwind; Integrated
        Security=SSPI");

    // 1. instantiate a new DataSet
    dsCustomers = new DataSet();
    // 2. init SqlDataAdapter with select
    command and connection
    daCustomers = new SqlDataAdapter(
        "select CustomerID, CompanyName from
        Customers", conn);
    // 3. fill in insert, update, and delete
    commands
    SqlCommandBuilder cmdBldr = new
    SqlCommandBuilder(daCustomers);

    // 4. fill the dataset
    daCustomers.Fill(dsCustomers, tableName);
}
// Update button was clicked
public void btnUpdateClicked(object sender,
EventArgs e)
{
    // write changes back to DataBase
    daCustomers.Update(dsCustomers,
tableName);
}
```

```
// start the Windows form
static void Main()
{
    Application.Run(new
DisconnectedDataForm());
}
}
```

SQL Basics

NOTES

For performance reasons, you will probably use the `DataSet` to work with a small subset of the total information in the data source. The tables in the `DataSet` do not need to map directly to tables in the data source. Using the `DataAdapter` it is possible to pull data out of one or multiple databases and to restore it with optimistic locking. The data and the structure from the database implemented by the `DataSet` class can be written to or read from XML documents and schemas.

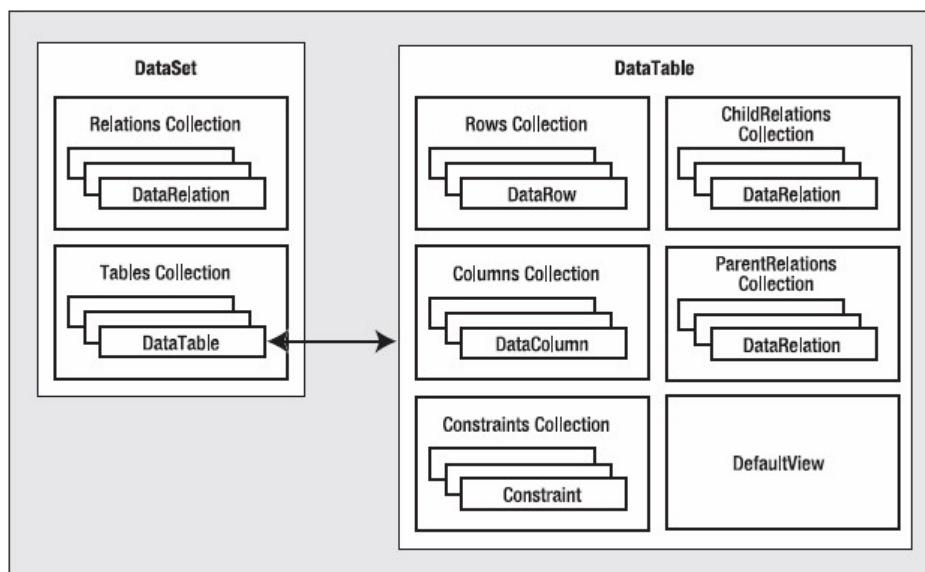


Fig. 14.2 Representation of DataSet and DataTable

As you can see in Figure 14.2, each record is represented as a `DataRow` object. To manage disconnected changes, the `DataSet` tracks versioning information for every `DataRow`. When you edit the value in a row, the original value is kept in memory and the row is marked as changed. When you add or delete a row, the row is marked as added or deleted.

Note: The data in the `DataSource` is not touched at all when you work with the `DataSet` objects. Instead, all the changes are made locally to the `DataSet` in memory.

The `DataSet` never retains any type of connection to a data source. If you want to extract records from a database and use them to fill a table in a `DataSet`, you need to use another ADO.NET object: a `DataAdapter`.

The DataAdapter also allows you to update the DataSource according to the changes you make in the DataSet. Table 14.1 shows the DataSet methods, while Table 14.2 shows DataSet properties.

NOTES

Table 14.1 DataSet Methods

Method	Description
Clear()	Empties all the data from the tables. However, this method leaves the schema and relationship information intact.
Copy()	Returns an exact duplicate of the DataSet, with the same set of tables, relationships, and data.
Clone()	Returns a DataSet with the same structure (tables and relationships) but no data.
Merge()	Takes another DataSet as input and merges it into the current DataSet, adding any new tables and merging any existing tables.

Table 14.2 DataSet Properties

Properties	Description
DataSetName	The name of this DataSet object.
Namespace	The namespace for this DataSet, used when importing or exporting XML data.
Tables	Returns the collection of child DataTable objects.
Relations	Returns the collection of DataRelation objects.
DefaultViewManager	Returns a DataViewManager object that allows you to create custom search and filter settings for the DataTable objects in the DataSet.

TypedDataSet

A TypedDataSet is a database object that fits in the object model of a .NET project. TypedDataSet generate classes that expose each object in the DataSet in Type-safe manner. These classes inherit directly from DataSet class.

Let us look into a small example which explains the TypedDataSet.

1. Using DataSet

```
//Create DataAdapter
SqlDataAdapter daEmp = new
SqlDataAdapter("SELECT
empno,empname,empaddress FROM
EMPLOYEE",conn);
//Create a DataSet Object
DataSet dsEmp = new DataSet();
//Fill the DataSet
daEmp.Fill(dsEmp,"EMPLOYEE");
```

```

//Let us print first row and first column of
the table
Console.WriteLine(dsEmp.Tables["EMPLOYEE"]
    .Rows[0][0].ToString());
//Assign a value to the first column
dsEmp.Tables["EMPLOYEE"].Rows[0][0] =
"12345";//This will generate runtime error
as empno column is integer

```

SQL Basics

NOTES

We will get a runtime error when this code gets executed as the value assigned to the column (empno) does not take string value.

2. Using Typed DataSet

```

//Create DataAdapter
SqlDataAdapter daEmp = new
SqlDataAdapter("SELECT
    empno, empname, empaddress FROM EMPLOYEE", conn);
//Create a DataSet Object
EmployeeDS dsEmp = new EmployeeDS();
//Fill the DataSet
daEmp.Fill(dsEmp, "EMPLOYEE");
//Let us print first row and first column of
the table
Console.WriteLine(dsEmp.EMPLOYEE[0].empno.ToString());
//Assign a value to the first column
dsEmp.EMPLOYEE[0].empno = "12345";//This will
generate compile time error.

```

In this code we find that a typed dataset is very much similar to a normal dataset. The only difference is that the schema is already present for the same. Hence, any mismatch in the column will generate compile time errors rather than runtime error as in the case of a normal dataset. A Typed DataSet can be generated using Visual Studio .NET IDE.

14.6 THE DataTable CLASS

A `DataTable` object resembles a database table and has its own collections—the `Columns` collection of `DataColumn` objects (which describe the name and data type of each field) and the `Rows` collection of `DataRow` object (which contain the actual data in each record). It can also have a primary key based on one or more columns and a collection of `Constraint` objects, which are useful for enforcing the uniqueness of the values in a column. `DataTable` objects in a `DataSet` class are often tied to each other through relationships, exactly as if they were database tables. Tables 14.3, 14.4 and 14.5 show `Datatable` properties, `Datatable` methods and `Datatable` events, respectively.

Table 14.3 Datatable Properties**NOTES**

Properties	Description
TableName	The name of this DataTable object.
Namespace	The namespace for this DataTable, used when importing or exporting XML data.
DataSet	Returns the DataSet this DataTable belongs to.
Rows	Returns the collection of child DataRow objects.
Columns	Returns the collection of child DataColumn objects.
Constraints	Returns the collection of the Constraint objects for this DataTable (for example, foreign key constraints or unique constraints).
PrimaryKey	An array of DataColumn objects that represent the primary keys for the DataTable.
DefaultView	Returns the DataView object that you can use to filter and sort this DataTable.
ChildRelations	Returns the collection of DataRelation objects in which this DataTable is the master table.
ParentRelations	Returns the collection of DataRelation objects in which this DataTable is the detail table.

Table 14.4 Datatable Methods

Method	Description
Clear()	Clears all the data in the DataTable.
Copy()	Creates a DataTable that has both the same structure and the same data as the current one.
Clone()	Creates a cloned DataTable that contains the identical structure, tables, and relationships as the current one.
AcceptChanges	Commits all changes to this DataTable after it was loaded or since the most recent AcceptChanges method.
RejectChanges	Rejects all changes to this DataTable after it was loaded or since the most recent AcceptChanges method.
Reset	Resets the DataTable to its original state.
Select	Returns the array of all the DataRow objects that satisfy the filter expression.

Table 14.5 Datatable Events

Events	Description
ColumnChanging	Fires when a DataColumn is changing. The event handler can inspect the new value.
ColumnChanged	Fires after a DataColumn have changed.
RowChanging	Fires when a DataRow is changing.
RowChanged	Fires after a DataRow have changed.
RowDeleting	Fires when a DataRow is being deleted.
RowDeleted	Fires after a DataRow have been deleted.

DataTable Class of C# in ASP.Net provides a Select function that allows you to filter the selective rows based on the search criteria passed to the select function. You can perform the rows search exactly similar to the SQL select query with WHERE clause. You just need to pass the column name along with conditional operator and value to filter the resulting rows based on the specified search criteria.

For example: “columnName = 10”, “columnName <= 10”, “columnName >= 10” or “columnName <> 10” for integer values, and “columnName = ‘Tim’” for string values.

```
// DataTable.Select function returns the
// DataRow collection based on the filterExpression
DataRow[] filterRows = myDataTable.Select(
    "Unit_Price < 40", "");
```

ASP.Net DataTable Select function accepts two types of parameters:

1. **FilterExpression:** Accepts the string type values as search string FilterExpression to filter the rows.
2. **Sort:** Accepts the string value as sort expression to sort the filtered rows based on the ColumnName and sort direction.

NOTES

14.6 THE DataRow CLASS

The DataRow class represents an individual row (or record) in a DataTable. Each DataRow contains one or more fields, which can be accessed through its Item property. Tables 14.6 and 14.7 show DataRow properties and methods, respectively.

Table 14.6 DataRow Properties

Properties	Description
Item	Gets or sets the data stored in the specified column. The argument can be the column name, the column index or a DataColumn object.
ItemArray	Gets or sets the values of all the columns, using an Object array.
RowState	The current state of this row can be Unchanged, Modified, Added, Deleted or Detached.
HasErrors	Returns True if there are errors in the column collection.
RowError	Gets or sets a string containing the custom error description for the current row.

Table 14.7 DataRow Methods**NOTES**

Methods	Description
AcceptChanges	Commits all changes to this DataRow after it was loaded or since the most recent AcceptChanges method.
RejectChanges	Rejects all changes to this DataRow after it was loaded or since the most recent AcceptChanges method.
BeginEdit	Marks the beginning of an edit operation on a DataRow.
EndEdit	Confirms all the changes to the DataRow since the most recent BeginEdit method.
CancelEdit	Cancels all the changes to the DataRow since the most recent BeginEdit method.
Delete	Deletes the row.
ClearErrors	Clear all errors for this row, including the RowError property and errors set with the SetColumnError method.
IsNull	Returns True if the specified column has a null value. The argument can be a column name, a column index or a DataColumn.
GetChildRows	Returns an array of the DataRow objects that are the child rows of the current row, following the relationship specified by the argument, which can be a relationship name or a DataRelation object.
GetParentRows	Returns an array of the parent DataRow objects, following the relationship specified by the argument.
SetParentRow	Sets the parent DataRow for the current row.

14.7 DataColumn CLASS

The DataColumn object represents a column in a DataTable. Add Method is used to add a data column to the DataColumnCollection. DataColumnCollection is a type represents a collection of columns attached to table and returns them using the Columns property of DataTable. Consider an example, you want to create an employee table having columns i.e. Emp_Id, Name and Address. You will create three DataColumn objects to DataColumnCollection using the DataTable.Column.Add method. Table 14.8 and 14.9 shows DataColumn Properties and methods respectively.

Table 14.8 DataColumn Properties

AllowDBNull	It gets or sets a value which indicates whether null values are allowed in the column for rows that belong to the table.
AutoIncrement	Gets or sets a value that indicates whether the column automatically increments the value of the column for new rows added to the table.
AutoIncrementSeed	Gets or sets the starting value for a column that has its AutoIncrement property set to true. The default is 0.
AutoIncrementStep	Gets or sets the increment used by a column with its AutoIncrement property set to true.

Caption	Gets or sets the caption for the column.	<i>SQL Basics</i>
ColumnMapping	Gets or sets the MappingType of the column.	
ColumnName	Gets or sets the name of the column in the DataColumn Collection.	
Container	Gets the container for the component. <i>(Inherited from MarshalByValueComponent)</i>	NOTES
DataType	Gets or sets the type of data stored in the column.	
DateTimeMode	Gets or sets the DateTimeMode for the column.	
DefaultValue	Gets or sets the default value for the column when you are creating new rows.	
DesignMode	Gets a value indicating whether the component is currently in design mode. <i>(Inherited from MarshalByValueComponent)</i>	
Events	Gets the list of event handlers that are attached to this component. <i>(Inherited from MarshalByValueComponent)</i>	
Expression	Gets or sets the expression used to filter rows, calculate the values in a column, or create an aggregate column.	
ExtendedProperties	Gets the collection of custom user information associated with a DataColumn.	
MaxLength	Gets or sets the maximum length of a text column.	
Namespace	Gets or sets the namespace of the DataColumn.	
Ordinal	Gets the (zero-based) position of the column in the DataColumn Collectioncollection.	
Prefix	Gets or sets an XML prefix that aliases the namespace of the DataTable.	
ReadOnly	Gets or sets a value that indicates whether the column allows for changes as soon as a row has been added to the table.	
Site	Gets or sets the site of the component. <i>(Inherited from MarshalByValueComponent)</i>	
Table	Gets the DataTable to which the column belongs to.	
Unique	Gets or sets a value that indicates whether the values in each row of the column must be unique.	

Table 14.9 *DataColumn Methods*

CheckNotNull ()	This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.
CheckUnique ()	This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.
Dispose ()	Releases all resources used by the MarshalByValue Component. <i>(Inherited from MarshalByValueComponent)</i>
Dispose (Boolean)	Releases the unmanaged resources used by the MarshalBy ValueComponentand optionally releases the managed resources. <i>(Inherited from MarshalByValueComponent)</i>

NOTES

<code>Equals (Object)</code>	Determines whether the specified object is equal to the current object. (Inherited from Object)
<code>GetHashCode ()</code>	Serves as the default hash function. (Inherited from Object)
<code>GetService (Type)</code>	Gets the implementer of the IServiceProvider. (Inherited from MarshalByValueComponent)
<code>GetType ()</code>	Gets the Type of the current instance. (Inherited from Object)
<code>MemberwiseClone ()</code>	Creates a shallow copy of the current Object. (Inherited from Object)
<code>OnPropertyChanging (PropertyChangedEventArgs)</code>	This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.
<code>RaisePropertyChanged (String)</code>	This member supports the .NET Framework infrastructure and is not intended to be used directly from your code.
<code>SetOrdinal (Int32)</code>	Changes the ordinal or position of the DataColumn to the specified ordinal or position.
<code>ToString ()</code>	Gets the Expression of the column, if one exists.

Check Your Progress

1. What is data components?
2. What is data package?
3. Write a note on DataSet.
4. What are the two collections of DataTable object?

14.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A data model is made up of components that represent real world sources of data and the actual data inside them. Data items are elements of the data model that represent actual units of data stored in a data source.
2. A Data Package is a simple way of putting collections of data and their descriptions in one place so that they can be easily shared and used.
3. A DataSet is an in-memory data store that can hold numerous tables. It contains two important ingredients: a collection of zero or more tables and a collection of zero or more relationships that you can use to link tables together. The tables in a DataSet are implemented by the DataTable class.
4. A DataTable object resembles a database table and has its own collections—the Columns collection of DataColumn objects (which

describe the name and data type of each field) and the Rows collection of DataRow object (which contain the actual data in each record).

SQL Basics

14.9 SUMMARY

NOTES

- Open the database connection in each and every method call, and close the connection before the method ends.
- The data component actually consists of two classes—a data package class that wraps a single record of information and a database utility class that performs the actual database operations with ADO.NET code.
- Disconnected data can be used to communicate between distributed applications or Web services. Like connected data, disconnected data supports multiple tables and data binding. In addition, it also supports forward and backward navigation in a result set.
- A stored procedure is an already written SQL statement that is saved in the database. Microsoft SQL Server provides the stored procedure mechanism to simplify the database development process by grouping Transact-SQL statements into manageable blocks.
- The CookieParameter class provides the CookieName property which identifies the name of the HttpCookie object to bind to, in addition to those inherited from the Parameter class.
- A DataSet is an in-memory data store that can hold numerous tables. It contains two important ingredients: a collection of zero or more tables and a collection of zero or more relationships that you can use to link tables together.
- The Sql Data Adapter holds the SQL commands and connection object for reading and writing data.
- A Typed Data Set is a database object that fits in the object model of a .NET project. Typed Data Set generate classes that expose each object in the DataSet in Type-safe manner.

14.10 KEY WORDS

- **Stored procedure:** It is an already written SQL statement that is saved in the database.
- **DataSet:** It is an in-memory data store that can hold numerous tables.
- **DataView:** It defines a view onto a DataTable object.

14.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES**Short Answer Questions**

1. What is the function of the `clear()` method of dataset?
2. Differentiate between `clone` and `copy` methods of dataset.
3. Write some of the properties and methods of `DataRow` Class.
4. Discuss the various properties and methods of `DataColumn` Class.

Long Answer Questions

1. Explain in detail the procedure to create and design a `Data` class.
2. Discuss the properties and description of various data components and `DataSet`.
3. Explain the properties of `DataSet` class and `DataTable` class.

14.12 FURTHER READINGS

- Reynolds, Matthew, Jonathan Crossland, Richard Blair and Thearon Willis . 2002. *Beginning VB.NET*, 2nd edition. Indianapolis: Wiley Publishing Inc.
- Cornell, Gary and Jonathan Morrison. 2001. *Programming VB .NET: A Guide for Experienced Programmers*, 1st edition. Berkeley: Apress.
- Francesc, Balena. 2002. *Programming Microsoft Visual Basic .NET*, 1st edition. United States: Microsoft Press.
- Liberty, Jesse. 2002. *Learning Visual Basic .NET*, 1st edition. Sebastopol: O'Reilly Media.
- Kurniawan, Budi and Ted Neward. 2002. *VB.NET Core Classes in a Nutshell*. Sebastopol: O'Reilly Media.