

Chapter 1

Introduction to .NET

1.0	Objectives
1.1	Introduction
1.2	.NET Framework
1.2.1	Net Framework and .Net Languages
1.2.2	Languages supported by .Net Framework
1.3	.NET Class Library
1.4	ASP vs. ASP.NET
1.5	Summary
1.6	Check your Progress - Answers
1.7	Questions for Self – Study
1.8	Suggested Readings

1.0 OBJECTIVES

After studying this chapter you will be able to –

- explain .NET Framework.
- describe the .Net languages.
- explain the .Net class library.
- explain difference between ASP and ASP.NET.

1.1 INTRODUCTION

Microsoft .NET technology you will have access to a new generation of advanced software joining the best of computing and communications in a revolutionary new way. The effect will be to totally transform the Web and every other aspect of the computing experience. .NET enables developers, businesses, and consumers to harness technology on their terms. .NET will allow the creation of truly distributed Web services that will integrate and collaborate with a range of complementary services to help customers in ways that today's dotcoms can only dream of.

The fundamental idea behind .NET is that the focus is shifting from individual Web sites or devices connected to the Internet to constellations of computers, devices, and services that work together to deliver broader, richer solutions. People will have control over how, when, and what information is delivered to them. Computers, devices and services will be able to collaborate with each other to provide rich services, instead of being isolated islands where the user provides the only integration. Businesses will be able to offer their products and services in a way that lets customers seamlessly embed them in their own electronic fabric.

Microsoft .NET will make computing and communicating simpler and easier than ever. It will spawn a new generation of Internet services, and enable tens of thousands of software developers to create revolutionary online services and businesses. It will put you back in control, and enable greater control of your privacy, digital identity, and data. And software is what makes it all possible. However, Microsoft's .NET technology will only succeed if others adopt this new standard.

Source: <http://vig.prenhall.com/samplechapter/013093285X.pdf>

1.2 .NET FRAMEWORK

The Microsoft .NET Framework is a software compatible work that is available with several Microsoft Windows operating systems. It includes a large library of pre-coded solutions to common programming problems and a virtual machine that

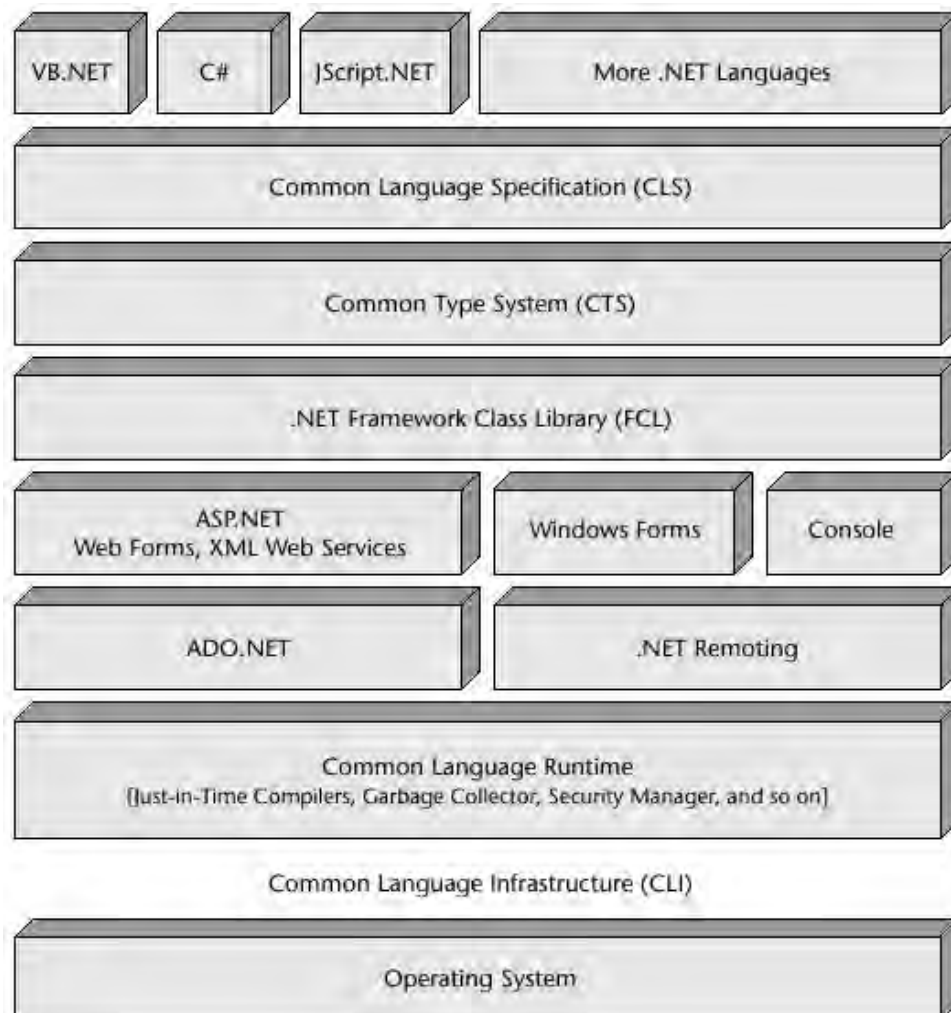
manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.

Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling.

Source: <https://sites.google.com/site/entirefacts/dotnet>

Version 3.0 of the .NET Framework is included with Windows Server 2008 and Windows Vista. The current version of the framework can also be installed on Windows XP and the Windows Server 2003 family of operating systems. A reduced "Compact" version of the .NET Framework is also available on Windows Mobile platforms, including smart phones.



NET Framework

Advantages of .NET

The .NET Framework provides the following advantages:

- A consistent, object-oriented programming environment.
- A code-execution environment that:
 - Promotes safe execution of code.

- Eliminates the performance problems of scripted or interpreted environments.
 - Minimizes software deployment and versioning conflicts.
- A consistent experience for both developers and users across various types of Windows-based and Web-based applications on multiple devices.
- Communication built on the industry standards to ensure that code based on the .NET Framework can integrate with any other code.

.NET is based on open Internet standards, which include Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), and Simple Object Access Protocol (SOAP).

1.2.1 .NET Framework and .NET Languages

As mentioned on the .NET Framework page, .NET Framework is designed for cross-language compatibility. Cross-language compatibility means .NET components can interact with each other irrespective of the languages they are written in. An application written in VB .NET can reference a DLL file written in C# or a C# application can refer to a resource written in VC++, etc. This language interoperability extends to Object-Oriented inheritance.

This cross-language compatibility is possible due to common language runtime. As you read on the .NET Framework page, when the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL). This MSIL is a low-level language which is designed to be read and understood by the common language runtime. Because all .NET executables exist as IL, they can freely operate. The Common Language Specification defines the minimum standards that .NET language compilers must conform to. Thus, any code compiled by a .NET compiler can interoperate with the .NET Framework.

The Common Type System (CTS) defines the rules concerning data types and ensures that code is executed in a safe environment. Since all .NET applications are converted to IL before execution all primitive data types are represented as .NET types. This means that, a VB Integer and a C# integer are both represented in IL code as System.Int32. Because both the languages use a common type system, it is possible to transfer data between components and avoid time-consuming conversions.

1.2.2 Languages supported by .NET Framework

The table below lists all the languages supported by the .NET Framework and describes those languages.

Language	Description/ Usage
APL	APL (A Programming Language) is one of the most powerful, consistent and concise computer programming languages ever devised. It is a language for describing procedures in the processing of information. It can be used to describe mathematical procedures having nothing to do with computers or to describe the way a computer works.
C++	C++ is a true OOP. It is one of the early Object-Oriented programming languages. C++ derives from the C language. Visual C++ is the name of a C++ compiler with an integrated environment from Microsoft. This includes special tools that simplify the development of great applications, as well as specific libraries. Its use is known as visual programming.
C#	C# called as C Sharp is a fully fledged Object-Oriented programming language from Microsoft built into the .NET Framework. First created in the late 1990's was part of Microsoft's whole .NET strategy.
Cobol	COBOL (Common Business Oriented Language) was the first widely-used high-level programming language for business applications. It is considered as a

	programming language to have more lines of code than any other language.
Eiffel	Eiffel is an Object-Oriented (OO) programming language which emphasizes the production of robust software. Eiffel is strongly statically typed mature Object-Oriented language with automatic memory management.
Fortran	Formula Translator, Fortran is one of the oldest high-level programming languages that are still widely used in scientific computing because of its compact notation for equations, ease in handling large arrays, and huge selection of library routines for solving mathematical problems efficiently.
Haskell	Haskell is a computer programming language that is a polymorphic typed, lazy, purely functional language, quite different from most other programming languages. It is a wide-spectrum language, suitable for a variety of applications. It is particularly suitable for programs which need to be highly modifiable and maintainable.
Java Language	The Java language is one of the most powerful Object-Oriented programming languages developed till date. Its platform independence (not depending on a particular OS) feature makes it a very popular programming language.
Microsoft JScript	Microsoft JScript is the Microsoft implementation of the ECMA 262 language specification. JScript is an interpreter, object-based scripting language. It has fewer capabilities than full-fledged Object-Oriented languages like C++ but is more than sufficiently powerful for its intended purposes.
Mercury	Mercury is a new logic/functional programming language, which combines the clarity and expressiveness of declarative programming with advanced static analysis and error detection features. Its highly optimized execution algorithm delivers efficiency far in excess of existing logic programming systems, and close to conventional programming systems. Mercury addresses the problems of large-scale program development, allowing modularity, separate compilation, and numerous optimization/time trades-offs.
Mondrian	Mondrian is a simple functional scripting language for Internet applications. It is a functional language specifically designed to inter-operate with other languages in an OO environment. Current versions of Mondrian run on .NET. Mondrian also supports ASP.NET, allowing you to embed functional language code in web pages along with C# code.
Oberon	Oberon is a programming language very much like Modula-2 in syntax but with several interesting features. It's based on OOP concepts and provides a Windows-based graphical user interface.
Oz	Oz is a high-level programming language that combines constraint inference with concurrency. Oz is dynamically typed and has first-class procedures, classes, objects, exceptions and sequential threads synchronizing over a constraint store. It supports finite domain and feature constraints and has powerful primitives for programming constraint inference engines at a high level.

Perl	Practical Extraction and Report Language, Perl, is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks.
Python	Python is an interpreter, interactive, Object-Oriented programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing.
RPG	Report Program Generator, RPG, is used for generation of reports from data files, including matching record and sub-total reports. RPG is one of the few languages created for punch card machines that are still in common use today. RPG or RPG IV is a native programming language for IBM's iSeries minicomputer system.
Scheme	Scheme is a statically scoped programming language. It was designed to have an exceptionally clear and simple semantics and few different ways to form expressions. A wide variety of programming paradigms, including imperative, functional, and message passing styles, find convenient expression in Scheme.
Small Talk	Small Talk is an expressive language that uses a simple sub set of human languages, nouns and verbs. Smalltalk was the first, and remains one of the few; pure object a system, which simply means that everything in a Smalltalk program is an object. Smalltalk is generally recognized as the second Object Programming Language (OPL).
Standard ML	Standard ML is a safe, modular, strict, functional, polymorphic programming language with compile-time type checking and type inference, garbage collection, exception handling, immutable data types and updatable references, abstract data types, and parametric modules. It has efficient implementations and a formal definition with a proof of soundness.
Microsoft Visual Basic	Visual Basic is a "visual programming" environment for developing Windows applications. Visual Basic makes it possible to develop complicated applications very quickly. This site is all about Visual Basic.

1.3 .NET CLASS LIBRARY

The .NET Framework class library is a library of classes, interfaces, and value types that are included in the Windows Software Development Kit (SDK). This library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built.

Namespaces:

Namespace	Description
System	The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
System.Data	The System.Data namespaces contain classes for accessing and managing data from diverse sources. The top-level namespace and a number of the child namespaces together form the ADO.NET architecture and ADO.NET data providers. For example, providers are available for SQL Server, Oracle, ODBC, and OleDb.

	Other child namespaces contain classes used by the ADO.NET Entity Data Model (EDM) and by WCF Data Services.
System.IO	The System.IO namespaces contain types that support input and output, including the ability to read and write data to streams either synchronously or asynchronously, to compress data in streams, to create and use isolated stores, to map files to an application's logical address space, to store multiple data objects in a single container, to communicate using anonymous or named pipes, to implement custom logging, and to handle the flow of data to and from serial ports.
System.Net	The System.Net namespaces contain classes that provide a simple programming interface for a number of network protocols, programmatically access and update configuration settings for the System.Net namespaces, define cache policies for web resources, compose and send e-mail, represent Multipurpose Internet Mail Exchange (MIME) headers, access network traffic data and network address information, and access peer-to-peer networking functionality. Additional child namespaces provide a managed implementation of the Windows Sockets (Winsock) interface and provide access to network streams for secure communications between hosts.
System.Web	The System.Web namespaces contain types that enable browser/server communication. Child namespaces include types that support ASP.NET forms authentication, application services, data caching on the server, ASP.NET application configuration, dynamic data, HTTP handlers, JSON serialization, incorporating AJAX functionality into ASP.NET, ASP.NET security, and web services.
System.Windows	The System.Windows namespaces contain types used in Windows Presentation Foundation (WPF) applications, including animation clients, user interface controls, data binding, and type conversion. System.Windows.Forms and its child namespaces are used for developing Windows Forms applications.

Source : from Microsoft MSDN

1.4 ASP VS. ASP.NET

ASP:

- 1) ASP is run under the inetinfo.exe (IIS) process space and hence susceptible to application crashes as a result the IIS needs to be stopped or restarted. ASP is related to the process isolation setting in IIS.
- 2) Classical ASP had no mechanism of running itself on non- Microsoft technology platforms like the 'The Apache Web Server'
- 3) In ASP only two languages were available for scripting VBScript and Jscript/JavaScript.
- 4) In ASP, ASP engine executes server-side code, which is always through an interpreter (JScript or VBScript). When a traditional ASP page is requested, the text of that page is parsed linearly. All content that is not server-side script is rendered as is back to the response. All server-side script in the page is first run through the appropriate interpreter (JScript or VBScript), the output of which is

then rendered back to the response. This architecture affects the efficiency of page rendering in several ways. Thus, to improve efficiency of rendering, many ASP developers resort to large blocks of server-side script, replacing static HTML elements with Response.Write() invocations instead. Finally, this ASP model actually allows different blocks of script within a page to be written in different script languages.

- 5) In classic ASP it was very difficult for us to debug the application. ASP developers had time to debug application due to limited support due to the interpreted model.
- 6) Class=docText>especially if you are using ASP pages it is possible to include executable code outside the scope of a function within a script block marked as runat=server, and, it is possible to define a function within a pair of server-side script tags.
- 7) New Page Directives: In ASP you must place all directives on the first line of a page within the same delimiting block. For example:

```
<%LANGUAGE="VBSCRIPT" CODEPAGE="932"%>
```

ASP.NET

- 1) The ASP.NET worker process is a distinct worker process, aspnet_wp.exe, separate from inetinfo.exe (IIS process), and the process model in ASP.NET is unrelated to process isolation settings in IIS.
- 2) ASP.NET could be run on non-Microsoft Platforms also.
- 3) We are no longer constrained to the two scripting languages available in traditional ASP: Any fully compliant .NET language can now be used with ASP.NET, including C# and VB.NET.
- 4) In contrast, ASP.NET pages are always compiled into .NET classes housed within assemblies. This class includes all of the server-side code and the static HTML, so once a page is accessed for the first time (or any page within a particular directory is accessed), subsequent rendering of that page is serviced by executing compiled code. This eliminates all the inefficiencies of the scripting model of traditional ASP. There is no longer any performance difference between compiled components and server-side code embedded within a page they are now both compiled components. There is also no performance difference between interspersing server-side code blocks among static HTML elements, and writing large blocks of server-side code and using Response .Write() for static HTML content. Also, because the .aspx file is parsed into a single code file and compiled, it is not possible to use multiple server-side languages within a single .aspx file.
- 5) But in ASP.NET In addition to improved performance over the interpreted model, pages that are compiled into classes can be debugged using the same debugging tools available to desktop applications or component developers. Errors with pages are generated as compiler errors, and there is a good chance that most errors will be found at compilation time instead of runtime, because VB.NET and C# are both strongly typed languages. Plus, all the tools available to the .NET developer are applicable to the .aspx developer.
- 6) In ASP.NET it is no longer possible to include executable code outside the scope of a function within a script block marked as runat=server, and conversely, it is no longer possible to define a function within a pair of server-side script tags.
- 7) But in ASP.NET, you are now required to place the Language directive with a Page directive, as follows:

```
<%@Page Language="VB" Code Page="932"%> <%@Output Cache  
Duration="60" Vary By Param="none" %>
```

You can have as many lines of directives as you need. Directives may be located anywhere in your .aspx file but standard practice is to place them at the beginning of the file. Several new directives have been added in ASP.NET.

1.5 SUMMARY

- The fundamental idea behind .NET is that the focus is shifting from individual Web sites or devices connected to the Internet to constellations of computers, devices, and services that work together to deliver broader, richer solutions.
- The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.
- .NET Framework is designed for cross-language compatibility. Cross-language compatibility means .NET components can interact with each other irrespective of the languages they are written in.
- All the languages supported by the .NET Framework is C++, C#, COBOL, Eiffel, Java, Perl etc.

Check your Progress – 1.2-1.4

True or False

- .Net Frame work is a software compatible work that is available with several Microsoft windows operating system.
- CLS defines rules concerning data types.
- C++ language is supported by .NET framework.
- MSIL is a low-level language.
- ASP.NET could not run on non-Microsoft Platforms.

1.6 CHECK YOUR PROGRESS – ANSWERS

- a. True b. False c. True d. True e. False

1.7 QUESTIONS FOR SELF - STUDY

- Explain brief introduction to .NET.
- Explain languages supported by .Net Frame work.
- Explain .NET framework with diagram.
- What is the difference between ASP and ASP.NET?

1.8 SUGGESTED READINGS

1. Microsoft MSDN
2. <http://vig.prenhall.com/samplechapter/013093285X.pdf>
3. Source: <http://devreminder.wordpress.com/net/net-framework-fundamentals/>
4. <https://sites.google.com/site/entirefacts/dotnet>
5. http://en.wikipedia.org/wiki/.NET_Framework_version_history



[illegible]

NOTES

[illegible]

C# Fundamentals

2.0	Objectives
2.1	Introduction
2.2	Writing Simple C# Program
2.3	Flow control
2.3.1	The if Statement
2.3.2	The Switch Statement
2.4	Boolean logic
2.5	The goto statement
2.6	Branching
2.7	Looping
2.7.1	Do Statement
2.7.2	For Statement
2.7.3	For Reach Statement
2.7.4	While Statement
2.8	OOP in C#
2.8.1	Constructors & Overloading
2.8.2	Namespace & Class Library
2.9	Inheritance
2.10	Summary
2.11	Check your Progress - Answers
2.12	Questions for Self – Study
2.13	Suggested Readings

2.0 OBJECTIVES

After studying this chapter you will be able to –

- ◆ Explain simple C# program.
- ◆ explain the flow control from C#
- ◆ explain the various statements like goto, branching, and looping
- ◆ explain OOP in C#

2.1 INTRODUCTION

Microsoft has also developed a brand new programming language C# (C Sharp). This language makes full use of .NET. It is a pure object oriented language. C# is a simple, modern object oriented and type-safe programming language derived from C and C++. It will immediately be familiar to C and C++ programmers. C# aims to combine the high productivity of visual basic and the raw power of C++.

Visual C# .NET is Microsoft's C# development tool. It includes an interactive development environment, visual designers for building windows and web applications, a compiler, and a debugger. Visual C# .NET is part of a suite of products; call Visual Studio .NET, which also included Visual Basic .NET, Visual C++ .NET, and the Jscript scripting language. All of these languages provide access to the Microsoft .NET Framework, which includes a common execution engine and a rich class library.

The modern design of C# eliminates the most common C++ programming errors. C# empowers the traditional C/C++ programmer by providing:

- Automated garbage collection that relieves the programmer of the burden of manual memory management
- Automatically initialized variables
- Type-safe variables

C# is a language that makes it far easier for developers to create and maintain applications that solve complex business problems.

2.2 WRITING SIMPLE C# PROGRAM

We'll now write the "Hello, world" example application to get to know the basic syntax and structure of writing C# applications. Program Hello.cs

```
using System;
class Hello
{
public static void Main(string[] args)
{
Console.WriteLine("Hello, world");
}
}
```

The default file extension for C# programs is .cs, as in hello.cs. The name of the program can be hello.cs or any name you want.

OUTPUT: Hello, world

Detailed description of this program is:

- The using System; directive references a namespace called System that is provided by the Microsoft .NET Framework class library. This namespace contains the Console class referred to in the Main method. Namespaces provide a hierarchical means of organizing the elements of one or more programs. A “using” directive enables unqualified use of the types that are members of the namespace. The “hello, world” program uses Console.WriteLine as shorthand for System.Console.WriteLine.
- The Main method is a member of the class Hello. It has the static modifier, and so it is a method on the class Hello rather than on instances of this class.
- The entry point for an application—the method that is called to begin execution—is always a static method named Main.
- The “hello, world” output is produced using a class library. The language does not itself provide a class library. Instead, it uses a class library that is also used by Visual Basic .NET and Visual C++ .NET.

Source : [http://msdn.microsoft.com/en-IN/library/aa664628\(v=vs.71\).aspx](http://msdn.microsoft.com/en-IN/library/aa664628(v=vs.71).aspx)

2.3 FLOW CONTROL

This section looks at the real nuts and bolts of the language: the statements that allow you to control the flow of your program rather than executing every line of code in the order it appears in the program.

Conditional Statements:

Conditional statements allow you to branch your code depending on whether certain conditions are met or on the value of an expression. C# has two constructs for branching code –if statement, which allows you to test whether a specific condition is met, and the switch statement, which allows you to compare an expression with a number of different values.

2.3.1 The if statement

The, if statement selects a statement for execution based on the value of a Boolean expression.

Example:

This example checks if the input character is lowercase, uppercase, or a number. Otherwise, it is not an alphanumeric character. The program makes use of the else-if ladder.

```
// statements_if_else2.cs
// else-if
using System;
public class IfTest
{
    static void Main (String[] args)
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();

        if (Char.IsUpper(c))
        {
            Console.WriteLine("Character is uppercase.");
        }
        else if (Char.IsLower(c))
        {
            Console.WriteLine("Character is lowercase.");
        }
        else if (Char.IsDigit(c))
        {
            Console.WriteLine("Character is a number.");
        }
        else
        {
            Console.WriteLine("Character is not alphanumeric.");
        }
    }
}
```

Input

ABC

Output

Enter a character: ABC

The character is uppercase.

2.3.2 The Switch Statement

The **switch** statement is a control statement that handles multiple selections and enumerations by passing control to one of the **case** statements within its body as the following example:

```
// statements_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch(n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
                cost += 25;
                goto case 1;
            case 3:
                cost += 50;
                goto case 1;
            default:
                Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
```

Input

2

Sample Output

Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.

Example 2:

```
// statements_switch2.cs
using System;
class SwitchTest
{
    static void Main()
    {
        int n = 2;
        switch(n)
        {
            case 1:
            case 2:
            case 3:
                Console.WriteLine("It's 1, 2, or 3.");
                break;
            default:
                Console.WriteLine("Not sure what it is.");
                break;
        }
    }
}
```

```

    }
}
}

```

Output

Its 1, 2 or 3

Check your progress. 2.3

Answer the following questions in 1-2 sentences.

a) What is C#?

.....

b) What is flow control?

.....

2.4 BOOLEAN LOGIC

The `bool` type represents Boolean logical quantities. The possible values of type `bool` are `true` and `false`.

No standard conversions exist between `bool` and other types. In particular, the `bool` type is distinct and separate from the integral types, and a `bool` value cannot be used in place of an integral value, and vice versa.

In the C and C++ languages, a zero integral value or a null pointer can be converted to the Boolean value `false`, and a non-zero integral value or a non-null pointer can be converted to the Boolean value `true`. In C#, such conversions are accomplished by explicitly comparing an integral value to zero or explicitly comparing an object reference to `null`.

C++ has a separate Boolean data type ('bool'), but with automatic conversions from scalar and pointer values that are very similar to those of C. This approach was adopted also by many later languages, especially by some scripting ones such as AWK. One problem with this approach is that the tests `if(t==TRUE){...}` and `if(t)` are not equivalent.

Boolean type; a `bool` value is either `true` or `false`

```
bool val1 = true;
```

```
bool val2 = false;
```

Examples:

```

using System;
class Test
{
    static void Main() {
        string s = "Test";
        string t = string.Copy(s);
        Console.WriteLine(s == t);
        Console.WriteLine((object)s == (object)t);
    }
}

```

Sample Output

```

True
False

```

2.5 THE GOTO STATEMENT

The **goto** statement transfers the program control directly to a labeled statement. A common use of **goto** is to transfer control to a specific switch-case label or the default label in a **switch** statement. The **goto** statement is also useful to get out of deeply nested loops.

Example:

```
// statements_goto_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch (n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
                cost += 25;
                goto case 1;
            case 3:
                cost += 50;
                goto case 1;
            default:
                Console.WriteLine("Invalid selection.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
```

Input

2

Sample Output

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

2.6 BRANCHING

Changing the flow of control in a program in response to some kind of input or calculated value is an essential part of a programming language. C# provides the ability to change the flow of control, either unconditionally, by jumping to a new location in the code, or conditionally, by performing a test.

Example :

```
using System;

class Program
{
    static void Main()
    {
        int x = 1;
```



```

int y = 1;

if (x == 1)
    Console.WriteLine("x == 1");
else
    Console.WriteLine("x != 1");

if (x == 1)
{
    if (y == 2)
    {
        Console.WriteLine("x == 1 and y == 2");
    }
    else
    {
        Console.WriteLine("x == 1 and y != 2");
    }
}
}
}

```

Output:

Student should evaluate this program for practice.

2.7 LOOPING

You can create loops by using the iteration statements. Iteration statements cause embedded statements to be executed a number of times, subject to the loop-termination criteria. These statements are executed in order, except when a jump statement is encountered.

The following keywords are used in iteration statements:

- [do](#)
- [for](#)
- [foreach](#)
- [while](#)

2.7.1 Do statement

The **do** statement executes a statement or a block of statements enclosed in `{ }` repeatedly until a specified expression evaluates to **false**. In the following example the **do** loop statements execute as long as the variable `y` is less than 5.

The do while construct consists of a process symbol and a condition. First, the code within the block is executed, and then the condition is evaluated. If the condition is true the code within the block is executed again. This repeats until the condition becomes false.

Example :

```

// statements_do.cs
using System;
public class TestDoWhile
{
    public static void Main ()
    {
        int x = 0;
        do
        {
            Console.WriteLine(x);
            x++;
        }
        while (x < 5);
    }
}

```

Output

0
1
2
3
4

2.7.2 For Statement

For loop executes a statement or a block of statements repeatedly until a specified expression evaluates to **false**. **For** loop is handy for iterating over arrays and for sequential processing. In the following example, the value of `int i` is written to the console and `i` is incremented each time through the loop by 1.

Example :

```
// statements_for.cs
// for loop
using System;
class ForLoopTest
{
    static void Main()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

Output

1
2
3
4
5

2.7.3 Foreach Statement

The **foreach** statement repeats a group of embedded statements for each element in an array or an object collection. The **foreach** statement is used to iterate through the collection to get the desired information, but should not be used to change the contents of the collection to avoid unpredictable side effects.

Example:

```
// cs_foreach.cs
class ForEachTest
{
    static void Main(string[] args)
    {
        int[] fibarray = new int[] { 0, 1, 2, 3, 5, 8, 13 };
        foreach (int i in fibarray)
        {
            System.Console.WriteLine(i);
        }
    }
}
```

Output

0
1
2
3
5
8
13

2.7.4 While Statement

The **while** statement executes a statement or a block of statements until a specified expression evaluates to **false**.

Example:

```
// statements_while.cs
using System;
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
```

Output

Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5

Check your progress. 2.7

Answer the following questions in 1-2 sentences.

a) What is Boolean logic?

.....
.....

b) Explain the goto statement.

.....
.....

c) Explain foreach statement.

.....
.....

2.8 OOP IN C#

What is OOP?

Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

All code and data in C# must be enclosed in a class. You can't define a variable outside of a class, and you can't write any code that's not in a class. Classes can have constructors, which execute when an object of the class is created, and a destructor, which executes when an object of the class is destroyed. Classes support

single inheritance, and all classes ultimately derive from a base class called object. C# supports versioning techniques to help your classes evolve over time while maintaining compatibility with code that uses earlier versions of your classes.

As an example, take a look at a class called Family. This class contains the two static fields that hold the first and last name of a family member as well as a method that returns the full name of the family member.

```
class Class1
{
    public string FirstName;
    public string LastName;
    public string FullName()
    {
        return FirstName + LastName;
    }
}
```

Note Single inheritance means that a C# class can inherit from only one base class.

C# enables you to group your classes into a collection of classes called a namespace. Namespaces have names, and can help organize collections of classes into logical groupings. As you begin to learn C#, it becomes apparent that all namespaces relevant to the .NET Framework begin with System. Microsoft has also chosen to include some classes that aid in backwards compatibility and API access. These classes are contained within the Microsoft namespace.

The most significant enhancement of the C++ language is its support for object-oriented programming (OOP). You will have to modify your approach to problem solving to derive all of the benefits of C++. For example, objects and their associated operations must be identified and all necessary classes and subclasses must be constructed.

2.8.1 Constructors and overloading

Often you'll want to have more than one function with the same name. The most common example of this is to have more than one constructor. In the examples shown so far, the constructor has taken a single parameter: a DateTime object. It would be convenient to be able to set new Time objects to an arbitrary time by passing in year, month, date, hour, minute, and second values. It would be even more convenient if some clients could use one constructor, and other clients could use the other constructor. Function overloading provides for exactly these contingencies.

The signature of a method is defined by its name and its parameter list. Two methods differ in their signatures if they have different names or different parameter lists. Parameter lists can differ by having different numbers or types of parameters. For example, in the following code the first method differs from the second in the number of parameters, and the second differs from the third in the types of parameters:

Source : Programming C#: Building .NET Applications with C#
By Jesse Liberty

```
void myMethod(int p1);
void myMethod(int p1, int p2);
void myMethod(int p1, string s1);
```

A class can have any number of methods, as long as each one's signature differs from that of all the others.

Example : illustrates our Time class with two constructors, one which takes a DateTime object, and the other which takes six integers.

Example : Overloading the constructor

```
public class Time
{
    // public accessor methods
    public void DisplayCurrentTime( )
    {
```

```

        System.Console.WriteLine("{0}/{1}/{2} {3}:{4}:{5}",
            Month, Date, Year, Hour, Minute, Second);
    }

    // constructors
    public Time(System.DateTime dt)
    {
        Year =    dt.Year;
        Month =   dt.Month;
        Date =    dt.Day;
        Hour =    dt.Hour;
        Minute =  dt.Minute;
        Second =  dt.Second;
    }

    public Time(int Year, int Month, int Date,
        int Hour, int Minute, int Second)
    {
        this.Year =    Year;
        this.Month =   Month;
        this.Date =    Date;
        this.Hour =    Hour;
        this.Minute =  Minute;
        this.Second =  Second;
    }

    // private member variables
    private int Year;
    private int Month;
    private int Date;
    private int Hour;
    private int Minute;
    private int Second;
}

public class Tester
{
    static void Main( )
    {
        System.DateTime currentTime = System.DateTime.Now;

        Time t = new Time(currentTime);
        t.DisplayCurrentTime( );
        Time t2 = new Time(2000,11,18,11,03,30);
        t2.DisplayCurrentTime( );

    }
}

```

Sample Output:

```

5/1/2002 8:53:05
11/18/2005 11:3:30

```

As you can see, the Time class in Example has two constructors. If a function's signature consisted only of the function name, the compiler would not know which constructors to call when constructing t1 and t2. However, because the signature includes the function argument types, the compiler is able to match the constructor call

for t1 with the constructor whose signature requires a DateTime object. Likewise, the compiler is able to associate the t2 constructor call with the constructor method whose signature specifies six integer arguments.

2.8.2 Namespaces and Class Library

Namespaces in the .NET Runtime are used to organize classes and other types into a single hierarchical structure. The proper use of namespaces will make classes easy to use and prevent collisions with classes written by other authors.

Namespaces can also be thought of as way to specify really long names for classes and other types without having to always type a full name.

Namespaces are defined using the namespace statement. For multiple levels of organization, namespaces can be nested:

```
namespace Outer
{
    namespace Inner
    {
        class MyClass
        {
            public static void Function() {}
        }
    }
}
```

That's a fair amount of typing and indenting, so it can be simplified by using the following instead:

```
namespace Outer.Inner
{
    class MyClass
    {
        public static void Function() {}
    }
}
```

Each source file can define as many different namespaces as needed.

Class library — refers to an assembly that can be used by other assemblies. Use of a class library does not cause the creation of a new application domain. Instead, a class library is loaded into the application domain that uses it. For instance, when an application uses a class library, that class library is loaded into the application domain for that application. If an application uses a class library A that itself uses a class library B, then both A and B are loaded into the application domain for the application.

2.8.3 Properties

A property is a member that provides access to a characteristic of an object or a class. Examples of properties include the length of a string, the size of a font, the caption of a window, the name of a customer, and so on. Properties are a natural extension of fields. Both are named members with associated types, and the syntax for accessing fields and properties is the same. However, unlike fields, properties do not denote storage locations. Instead, properties have accessors that specify the statements to be executed when their values are read or written. Properties are defined with property declarations. The first part of a property declaration looks quite similar to a field declaration. The second part includes a get accessor and/or a set accessor. In the example below, the Button class defines a Caption property.

```
public class Button
{
    private string caption;
    public string Caption {
        get {
            return caption;
        }
        set {
            caption = value;
        }
    }
}
```

```

    Repaint();
}
}
...
}

```

Properties that can be both read and written, such as `Caption`, include both get and set accessors. The get accessor is called when the property's value is read; the set accessor is called when the property's value is written. In a set accessor, the new value for the property is made available via an implicit parameter named `value`.

2.9 INHERITANCE

In C#, the specialization relationship is typically implemented using inheritance. This is not the only way to implement specialization, but it is the most common and most natural way to implement this relationship. Saying that `ListBox` inherits from (or derives from) `Window` indicates that it specializes `Window`. `Window` is referred to as the base class, and `ListBox` is referred to as the derived class. That is, `ListBox` derives its characteristics and behaviors from `Window` and then specializes to its own particular needs.

Implementing Inheritance : In C#, you can create a derived class by adding a colon after the name of the derived class, followed by the name of the base class: `public class ListBox : Window`. This code declares a new class, `ListBox`, which derived from `Window`. You can read the colon as "derives from." The derived class inherits all the members of the base class, both member variables and methods. The derived class is free to implement its own version of a base class method. It does so by marking the new method with the keyword `new`.

Example ∴ Using a derived class

```

using System;

public class Window
{
    // constructor takes two integers to
    // fix location on the console
    public Window(int top, int left)
    {
        this.top = top;
        this.left = left;
    }

    // simulates drawing the window
    public void DrawWindow( )
    {
        Console.WriteLine("Drawing Window at {0}, {1}",
            top, left);
    }

    // these members are private and thus invisible
    // to derived class methods; we'll examine the
    // later in the chapter
    private int top;
    private int left;
}

// ListBox derives from Window
public class ListBox : Window
{
    // constructor adds a parameter
    public ListBox(
        int top,
        int left,
        string theContents):

```

```

        base(top, left) // call base constructor
    {
        mListBoxContents = theContents;
    }

    // a new version (note keyword) because in the
    // derived method we change the behavior
    public new void DrawWindow( )
    {
        base.DrawWindow( ); // invoke the base method
        Console.WriteLine ("Writing string to the listbox: {0}",
            mListBoxContents);
    }
    private string mListBoxContents; // new member variable
}

public class Tester
{
    public static void Main( )
    {
        // create a base instance
        Window w = new Window(5,10);
        w.DrawWindow( );

        // create a derived instance
        ListBox lb = new ListBox(20,30,"Hello world");
        lb.DrawWindow( );
    }
}

```

Output:

Drawing Window at 5, 10
 Drawing Window at 20, 30
 Writing string to the listbox: Hello world

Source : Learning C# 2005 by Jesse Liberty

Check your progress 2.9

Answer the following questions.

a) What is OOP?

.....

b) Define inheritance.

.....

2.10 SUMMARY

- ◆ C# is a simple, modern object oriented and type-safe programming language derived from C and C++.
- ◆ Conditional statements allow you to branch your code depending on whether certain conditions are met or on the value of an expression.

- ◆ The bool type represents Boolean logical quantities. The possible values of type bool are true and false.
- ◆ All code and data in C# must be enclosed in a class. Classes can have constructors, which execute when an object of the class is created, and a destructor, which executes when an object of the class is destroyed. Classes support single inheritance, and all classes ultimately derive from a base class called object.

2.11 CHECK YOUR PROGRESS – ANSWERS

2.3

- a. C# is a pure object oriented language, derived from C, C++ and developed by Microsoft for DOT NET compliance.
- b. Flow control is a mechanism where flow of program directed on basis of condition.

2.7

- a. Boolean logic depends upon two values “True or False” and works for true not for false. Ex. If statement.
- b. Goto statement is control transfer statement. It can transfer the control forward or back to a particular label name.
- c. For each statement works continuously on group of statement. It is different from for loop, because we repeat in groups by a variable which goes forward automatically.

2.9

- a. OOP stands for Object Oriented Programming Language. It is different from traditional procedural Language by creating class and object with some other features.
- b. Inheritance concept belongs to reusability of class. That means if we have created a class with some features and again we want that features in another class, so we just inherit later class from previous class.

2.12 QUESTIONS FOR SELF - STUDY

Long Answer Questions

1. Explain C# features with example.
2. Explain flow control with example.
3. What is goto statement? Explain with example.
4. Explain Branching with program.
5. What is loop? Explain with example.
6. Explain OOP in C#.
7. Explain Inheritance with example.

2.13 SUGGESTED READINGS

1. Programming C#: Building .NET Applications with C# By Jesse Liberty
2. Learning C# 2005 by Jesse Liberty Microsoft MSDN



NOTES

This image shows a single page of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting or typing. There are no margins, text, or other markings on the page.

Introduction to ASP.NET 2.0

3.0	Objectives
3.1	Introduction
3.2	Features of ASP.net 2.0
3.3	Stages in Web Forms Processing
3.4	Introduction to Server Controls
3.5	HTML Controls
3.6	Validation Controls
3.7	User Control
3.8	Data Binding Controls
3.9	Configuration
3.10	Personalization
3.10.1	User profiles
3.10.2	Personalization Providers
3.10.3	Using personalization
3.10.4	Defining profiles in Web .Config
3.10.5	User Profile information
3.10.6	Saving Profile Changes
3.10.7	Profiles and Users
3.11	Session State
3.12	Summary
3.13	Check your Progress - Answers
3.14	Questions for Self – Study
3.15	Suggested Readings

3.0 OBJECTIVES

After studying this Chapter You will be able to –

- ♦ explain the features of ASP.NET 2.0
- ♦ discuss how Web Forms pages work in Web applications
- ♦ discuss the HTML server controls with validation.
- ♦ state Data Binding control in .NET
- ♦ describe configuration and personalization?

3.1 INTRODUCTION

ASP.NET is a server-side Web application framework designed for Web development to produce dynamic Web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.

ASP.NET is the latest version of Microsoft's Active Server Pages technology (ASP).

It is the next generation ASP, but it's not an upgraded version of ASP. ASP.NET is an entirely new technology for server-side scripting. It was written from the

ground up and is not backward compatible with classic ASP. ASP.NET is the major part of the Microsoft's .NET Framework.

Before you continue you should have a basic understanding of the following: WWW, HTML, XML and the basics of building Web pages scripting languages like JavaScript or VBScript the basics of server side scripting like ASP or PHP

What is ASP.NET?

ASP.NET is a server side scripting technology that enables scripts (embedded in web pages) to be executed by an Internet server.

- Microsoft Technology
- ASP.NET is a program that runs inside IIS
- IIS (Internet Information Services) is Microsoft's Internet server
- IIS comes as a free component with Windows servers
- IIS is also a part of Windows 2000 and XP Professional

What is an ASP.NET File?

- An ASP.NET file is just the same as an HTML file
- An ASP.NET file can contain HTML, XML, and scripts
- Scripts in an ASP.NET file are executed on the server
- An ASP.NET file has the file extension ".aspx"

How Does ASP.NET Work?

- When a browser requests an HTML file, the server returns the file
- When a browser requests an ASP.NET file, IIS passes the request to the ASP.NET engine on the server
- The ASP.NET engine reads the file, line by line, and executes the scripts in the file
- Finally, the ASP.NET file is returned to the browser as plain HTML

3.2 FEATURES OF ASP.NET 2.0

ASP.NET 2.0 improves ASP.NET by adding several new features.

Design goals for ASP.NET 2.0:

- Increase productivity by removing 70% of the code
- Use the same controls for all types of devices
- Provide a faster and better web server platform
- Simplify compilation and installation
- Simplify the administration of web applications

Some of the new features in ASP.NET 2.0 are:

- Master Pages, Themes, and Web Parts
- Standard controls for navigation
- Standard controls for security
- Roles, personalization and internationalization services
- Improved and simplified data access controls

- Full support for XML standards like, XHTML, XML, and WSDL
- Improved compilation and deployment (installation)
- Improved site management
- New and improved development tools

The new features are described below.

- 1) **Master Pages** : ASP.NET didn't have a method for applying a consistent look and feel for a whole web site. **Master pages** in ASP.NET 2.0 solve this problem. A master page is a template for other pages, with shared layout and functionality. The master page defines placeholders for content pages. The result page is a combination (merge) of the master page and the content page.

Master Page Example:

```
<%@ Master %>
<html>
<body>
<h1>Standard Header For All Pages</h1>
<asp:ContentPlaceHolder id="CPH1" runat="server">
</asp:ContentPlaceHolder>
</body>
</html>
```

- 2) **Themes** : Themes are another feature of ASP.NET 2.0. Themes, or skins, allow developers to create a customized look for web applications.

Design goals for ASP.NET 2.0 themes:

- Make it simple to customize the appearance of a site
- Allow themes to be applied to controls, pages, and entire sites
- Allow all visual elements to be customized

- 3) **Web Parts** : ASP.NET 2.0 Web Parts can provide a consistent look for a site, while still allowing user customization of style and content.

New controls:

- Zone controls - areas on a page where the content is consistent
- Web part controls - content areas for each zone

- 4) **Navigation** : ASP.NET 2.0 has built-in navigation controls like

- Site Maps
- Dynamic HTML menus
- Tree Views

- 5) **Security** : Security is very important for protecting confidential and personal information. In ASP.NET 2.0 the following controls has been added:

- A Login control, which provides login functionality
- A LoginStatus control, to control the login status
- A LoginName control to display the current user name
- A LoginView control, to provide different views depending on login status
- A CreateUser wizard, to allow creation of user accounts
- A PasswordRecovery control, to provide the "I forgot my password" functionality

- 6) **Roles and Personalization**: Internet communities are growing very popular. ASP.NET 2.0 has personalization features for storing user details. This provides an easy way to customize user (and user group) properties.

- 7) **Internationalization:** Reaching people with different languages is important if you want to reach a larger audience. ASP.NET 2.0 has improved support for multiple languages.
- 8) **Data Access:** Many web sites are data driven, using databases or XML files as data sources. With ASP.NET this involved code, and often the same code had to be used over and over in different web pages. A key goal of ASP.NET 2.0 was to ease the use of data sources. ASP.NET 2.0 has new data controls, removing much of the need for programming and in-depth knowledge of data connections.
- 9) **Mobility Support:** The problem with Mobile devices is screen size and display capabilities. In ASP.NET, the Microsoft Mobile Internet Toolkit (MMIT) provided this support. In ASP.NET 2.0, MMIT is no longer needed because mobile support is built into all controls.
- 10) **Images:** ASP.NET 2.0 has new controls for handling images:
 - The ImageMap control - image map support
 - The DynamicImage control - image support for different browsers

These controls are important for better image display on mobile devices, like hand-held computers and cell phones.

- 11) **Automatic Compilation:** ASP.NET 2.0 provides automatic compilation. All files within a directory will be compiled on the first run, including support for WSDL, and XSD files.
- 12) **Compiled Deployment (Installation) and Source Protection:** ASP.NET 2.0 also provides pre-compilation. An entire web site can be pre-compiled. This provides an easy way to deploy (upload to a server) compiled applications, and because only compiled files are deployed, the source code is protected.
- 13) **Site Management:** ASP.NET 2.0 has three new features for web site configuration and management:
 - New local management console
 - New programmable management functions (API)
 - New web-based management tool
- 14) **Development Tools:** With ASP.NET Visual Studio.NET was released with project and design features targeted at corporate developers. With ASP.NET 2.0, Visual Studio 2005 was released. Key design features for Visual Studio 2005 include:
 - Support for the features described above
 - Upload files from anywhere (FTP, File System, Front Page....)
 - No project files, allowing code to be manipulated outside Visual Studio
 - Integrated Web Site Administration Tool
 - No "build" step - ability to compile on first run

Visual Web Developer is a new free ASP.NET 2.0 tool for non-corporate developers who don't have access to Visual Studio.NET.

Check your progress 3.2

Answer the following questions in 1-2 sentences.

a) What is ASP.NET?

.....
.....

b) Write any two ASP.NET features.

.....
.....

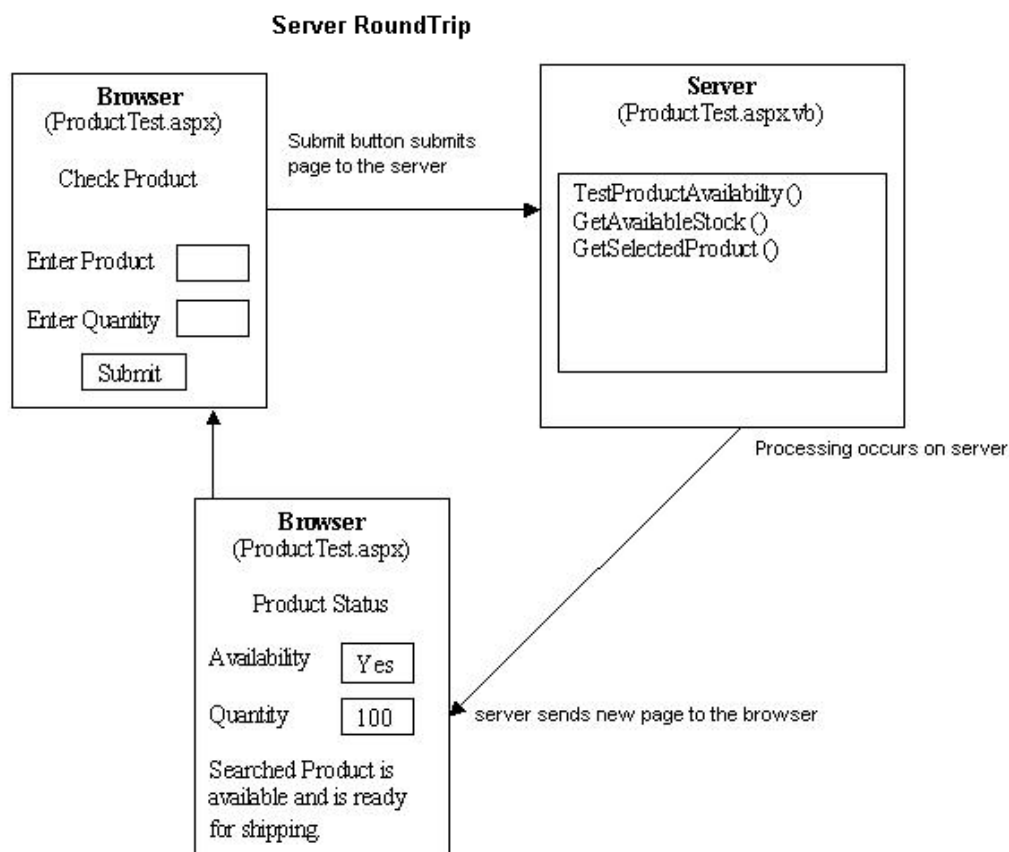
3.3 STAGES IN WEB FORMS PROCESSING

Web Forms Page Life Cycle: In general, the life cycle for a Web Forms page is similar to that of any Web process that runs on the server. Certain characteristics of Web processing information is passed via HTTP protocol, the stateless nature of Web pages, and so on. However, the ASP.NET page framework performs many Web application services for us. It is important to understand the sequence of events that occur when a Web Forms page is processed.

The Life Cycle of a Web Forms Page: It will be helpful to understand some fundamental characteristics of how Web Forms pages work in Web applications before we examine the details of what goes on inside a page when it is processed.

Round Trips : Most Web pages require processing on the server. For example, consider a products page used to check the availability of a certain product. When a user selects his product and hits the submit button the page must check on the server to see whether the selected product is available or not. This kind of functionality is achieved by handling server control events. Whenever a user interaction requires processing on the server, the Web page is posted back to the server processed and is returned back to the browser. This sequence is called round trip.

The image below demonstrates server round trip.



In any Web scenario, Web pages are recreated with every round trip. When the server finishes processing and sends the page to the browser, it discards the page information. This frees server resources after each request and a Web application can scale to support hundreds or thousands of simultaneous users. The next time the page is posted, the server starts over in creating and processing it, and for this reason, Web pages are said to be stateless. Stateless means the values of a page's variables and controls are not saved on the server.

Limitations of ASP.NET are as follows:

- ASP.NET saves page and control properties between round trips. This is referred to as saving the view state of the control.
- It provides state management facilities so that you can save your own variable and application-specific or session-specific information between round trips.
- It can detect when a form is requested for the first time versus when the form is posted, and allows you to program accordingly. You may want a different behavior during a page postback versus an initial request.

Stages in Web Forms Processing:

Stage	Means	Use
Page Initialization	The page's Page_Init event is raised, and the page and control view state are restored.	During this event, the ASP.NET page framework restores the control properties and postback data.
User Code Initialization	The page's Page_Load event is raised.	Read and restore values stored previously, Using the Page.IsPostBack property, check whether this is the first time the page is being processed. If this is the first time the page is being processed then perform initial data binding. Otherwise, restore control values. Read and update control properties.
Validation	The Validate method of any validator <u>Web server</u> controls is invoked to perform the control's specified validation.	Test the outcome of validation in an event handler
Event Handling	If the page was called in response to a form event, the corresponding event handler in the page is called during this stage	Perform application-specific processing and handle the specific event raised.
Cleanup	The Page_Unload event is called because the page has finished rendering and is ready to be discarded.	Perform final cleanup work. Close files, closing <u>database</u> Connections and discard objects.

Check your progress 3.3

Answer the following questions in 1-2 sentences.

a) What is Web form page life cycle?

.....

b) State the stages of Web form processing.

.....

3.4 INTRODUCTION TO SERVER CONTROLS

For creating Web Forms pages, you can use these types of controls:

- **HTML server controls** HTML elements exposed to the server so you can program them. HTML server controls expose an object model that maps very closely to the HTML elements that they render.
- **Web server controls** with more built-in features than HTML server controls. Web server controls include not only form-type controls such as buttons and text boxes, but also special-purpose controls such as a calendar. Web server controls are more abstract than HTML server controls in that their object model does not necessarily reflect HTML syntax.
- **Validation controls** that incorporate logic to allow you to test a user's input. You attach a validation control to an input control to test what the user enters for that input control. Validation controls are provided to allow you to check for a required field, to test against a specific value or pattern of characters, to verify that a value lies within a range, and so on.
- **User controls** that you create as Web Forms pages. You can embed Web Forms user controls in other Web Forms pages, which is an easy way to create menus, toolbars, and other reusable elements.

Every server control has a tag with the *runat="server"* attribute and is converted to client-specific XHTML. ASP.NET server controls are objects and have their own properties, methods and event handlers. Interesting to know is that server controls can keep their state (value and design) stored between subsequent page requests thanks to ViewState.

Every control is derived from the Control class and inherits the properties and the methods of this Control class. The most useful inherited properties are:

- Controls: gives a collection with all the child controls for a specified server control
- EnableViewState: Boolean to decide whether the control has to track his current state or not
- ID: id of the server control, maybe the most important property
- Visible: Boolean to decide if the control will be rendered

The most useful inherited methods are:

- DataBind(): binds the control with a certain data source (e.g. resultset of an sql query)
- FindControl(): searches a child control with the given name (string)
- HasControls(): Booleans that defines if this server control owns child controls

1) Page in ASP.NET

It might sound strange but the whole page is a control itself and acts as container for everything that is placed in it. Some of the properties are: Title, Session, Cache, Trace, Special on the Page control are the directives (start with a @). In the Page directive you find what language is used for the code behind, the path to the source file and the name of the class in the code behind and possible extra properties for the page document.

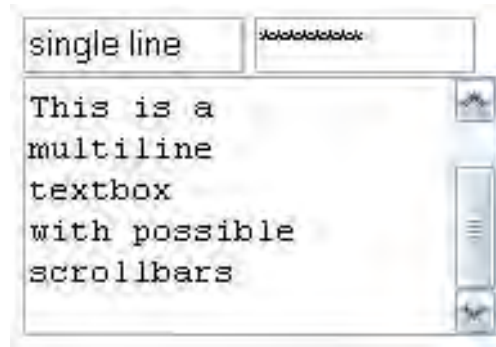
2) Label Control

The Label control displays text in a set location on the page. Unlike static text, the *Text* property of a label can be changed programmatically. Layout of the text can be done programmatically by changing properties or with HTML tags (which are allowed) in the text itself.

3) TextBox Control

The TextBox control enables the user to enter text. You have the possibility to use the textbox as a single line, multiline or password field by changing the *TextMode* property. The display width of a textbox is determined by its *Columns* property and the height of a textbox is determined by the *Rows* property (only for MultiLine textmode). You can limit the number of signs the user can enter for a textbox with its textmode set to SingleLine or Password by entering a value for *MaxLength*.

A very useful event for the TextBox control is *OnTextChanged*. When the cursor focus leaves the textbox this event is triggered. You can use this event to control the input at the very moment the user is done with editing his text.



```
<form id="form1" runat="server">
<div>
<br />
<br />
<asp:TextBox ID="txtSingleLine" runat="server" Columns="10"
MaxLength="40"></asp:TextBox>
<asp:TextBox ID="Password" runat="server" Columns="10"
TextMode="Password"></asp:TextBox>
<br />
<asp:TextBox ID="txtMultiLine" runat="server" Columns="10" Rows="5"
TextMode="Multiline">
</asp:TextBox>
</div>
</form>
```

4) Button Control

The Button control is used to submit the form back to the server (postback). This happens automatically in opposite to other controls where you have to enable the *AutoPostBack* property to have the same effect. Buttons have a *CausesValidation* property to trigger validation of all validation controls assigned to the same group.

It is possible to use several buttons in a container control like a DataGrid or a GridView. All these buttons have the same ID and it will be impossible to bind an event handler to each button control's *OnClick* event (or other event) to determine the particular button that was clicked. That's why buttons have a *CommandName* and *CommandArgument* property to determine which button you clicked and deliver the correct arguments to the event handler.

There are three types of buttons: the standard button, the *ImageButton* (image acting as a button) and the *LinkButton* (link acting as a button). Notice that the *ImageButton* doesn't have a *Text* property, but instead it does have an *AlternateText* property which shows the entered text if the browser doesn't support images.



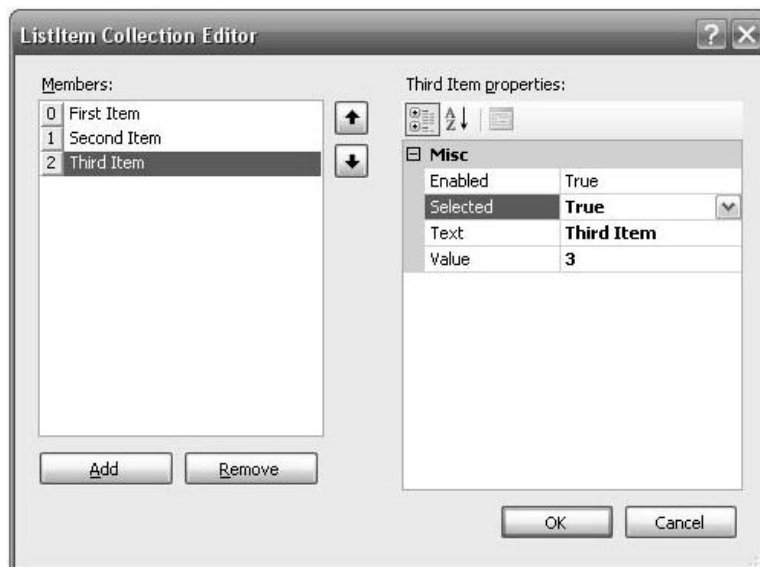
```
<asp:Button ID="btnButton" runat="Server" Text="Button" />
<asp:ImageButton ID="imgImageButton" runat="Server" ImageUrl="image.jpg" />
<asp:LinkButton ID="lnkLinkButton" runat="Server">Link Button</asp:LinkButton>
```

The most important event for a Button control is the *OnClick* event. This event is triggered with a click on the button and submits the form to the server. The *OnCommand* event is also triggered with a click on the button, but the values of *CommandName* and *CommandArgument* are also passed to the event.

5) DropDownList Control

The DropDownList control provides a single-select dropdown list. Each list item has a value and an index. You can add items to the list one by one yourself or by data binding a static array or another data source (e.g. results from a sql query). To bind a data source you'll be using the *DataTextField* property to link the text content and the *DataValueField* property to link the value of each list item.

There are several ways to add items to the list at Design time. In design view you can use the smart tag '*Edit items*' which shows on the dropdown list or you can click on the "..." next to the property *Items*. Both give the same window in which you can add the items one by one. Or you can write the code for each list item in the source view.



```
<asp:DropDownList ID="ddlDropDownList" runat="Server">
  <asp:ListItem Value="1">First Item</asp:ListItem>
  <asp:ListItem Value="2">Second Item</asp:ListItem>
  <asp:ListItem Selected="True" Value="1">Third Item</asp:ListItem>
</asp:DropDownList>
```

Adding items is not only limited to design time but can be done during run-time too. The best place to do this is in the Page_Load so you're sure everything is in your dropdown list before it is shown. It doesn't matter if you want to add them one by one or by data binding, both work fine.

```
if (!Page.IsPostBack) {
    // adding these one by one yourself
    ddlBackColor.Items.Add(new ListItem("red", "1"));
    ddlBackColor.Items.Add(new ListItem("green", "2"));
    ddlBackColor.Items.Add(new ListItem("blue", "3"));
    ddlBackColor.Items.Add(new ListItem("yellow", "4"));

    // adding these to an ArrayList and databinding the ArrayList
    // note this could have been replaced with results from a sql query
    ArrayList fontList = new ArrayList();
    fontList.Add("Arial");
    fontList.Add("Courier");
    ddlFont.DataSource = fontList;
    ddlFont.DataBind();
}
```

You can use the *OnSelectedIndexChanged* event to dynamically change layout/content of your web page when the user chooses another item in the list. But don't forget to put the *AutoPostBack* property on true if you want this to happen automatically without having to click on a button.

6) CheckBox(List) Control

The CheckBox control accepts Boolean input and stores it in the *Checked* property: true if the box is selected or false if it's not selected. Typically a checkbox is processed as one of several fields in a form. The CheckBoxList control provides a multiple-selection checked list. Like other list controls, CheckBoxList has an Items collection that corresponds to each item (CheckBox) in the list. You can test the *Selected* property of each item. You can choose to render the list in with or without table structure by the *RepeatLayout* property. Next to that you can also choose to render the list horizontally or vertically by changing the *RepeatDirection* property.

7) RadioButton(List) Control

The RadioButton control permits you to intersperse the radio buttons in a group with other content on the page. Be sure you group the RadioButtons logically by giving them all the same *GroupName* or you'll be able to select more than one.

If you tend to forget grouping the radio buttons, it might be better to use RadioButtonList. The RadioButtonList control provides a single-selection checked list. You can change the rendering like you did with a CheckBoxList.

8) Panel Control

The panel control is mainly used as a container for other controls. It's useful to group, hide or generate controls programmatically. If you generate controls programmatically you'll have to recreate them every time you load the page since the ViewState only remembers the state and not the controls you generated. If you only want to generate controls you can replace the panel by a *Placeholder*.

3.5 HTML CONTROLS

HTML elements in ASP.NET files are, by default, treated as text. To make these elements programmable, add a `runat="server"` attribute to the HTML element. This attribute indicates that the element should be treated as a server control.

Note: ASP.NET requires that all HTML elements must be properly closed and properly nested.

HTML Server Control	Description
HtmlAnchor	Controls an <a> HTML element
HtmlButton	Controls a <button> HTML element
HtmlForm	Controls a <form> HTML element
HtmlGeneric	Controls other HTML element not specified by a specific HTML server control, like <body>, <div>, , etc.
HtmlImage	Controls an <image> HTML element
HtmlInputButton	Controls <input type="button">, <input type="submit">, and <input type="reset"> HTML elements
HtmlInputCheckBox	Controls an <input type="checkbox"> HTML element
HtmlInputFile	Controls an <input type="file"> HTML element
HtmlInputHidden	Controls an <input type="hidden"> HTML element
HtmlInputImage	Controls an <input type="image"> HTML element
HtmlInputRadioButton	Controls an <input type="radio"> HTML element
HtmlInputText	Controls <input type="text"> and <input type="password"> HTML elements
HtmlSelect	Controls a <select> HTML element
HtmlTable	Controls a <table> HTML element
HtmlTableCell	Controls <td> and <th> HTML elements
HtmlTableRow	Controls a <tr> HTML element
HtmlTextArea	Controls a <textarea> HTML element

1) Example of HTMLButton

```
<script runat="server">
Sub button1(Source As Object, e As EventArgs)
    p1.InnerHtml="You clicked the blue button!"
Sub button2(Source As Object, e As EventArgs)
    p1.InnerHtml="You clicked the pink button!"
End Sub
</script>

<html>
<body>

<form runat="server">
<button id="b1" OnServerClick="button1"
style="background-color:#e6e6fa;
height=25;width:100" runat="server">
Blue button!
</button>
<button id="b2"
OnServerClick="button2"
style="background-color:#fff0f5;
height=25;width:100" runat="server">
Pink button!
</button>
<p id="p1" runat="server" />
</form>

</body>
</html>
```

Output Result

Blue button!

Pink button!

You clicked the blue button!

2) Example of HTMLForm

```
<script language = "vb" runat= "server">
if name.value<>" " then
    p1.InnerHtml="Welcome " & name.value & "!"
end if
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name: <input id="name" type="text" size="30" runat="server" />
<br /><br />
<input type="submit" value="Submit" OnServerClick="submit" runat="server" />
<p id="p1" runat="server" />
</form>

</body>
</html>
```

Output Result

Enter your name:

Welcome Hi Friends!

3) Example of [HtmlInputCheckBox](#)

```
<script language = "vb" runat="server">
if red.Checked=True then
    p1.InnerHtml="You prefer red!"
else
    p1.InnerHtml="You prefer blue!"
end if
red.checked=false
blue.checked=false
End Sub
</script>

<html>
<body>

<form runat="server">
What color do you prefer?
<br />
<input id="red" type="checkbox" runat="server" /> Red
<br />
<input id="blue" type="checkbox" runat="server" /> Blue
<br />
<input type="button" value="Submit" OnServerClick="submit" runat="server"/>
<p id="p1" runat="server" />
</form>

</body>
</html>
```

Output Result

What color do you prefer?

☒ Red

☐ Blue

You prefer red!

3.6 VALIDATION CONTROLS

Before saving the data entered by the user into a webform, we will have to validate the data. A web form is form which is used to accept data from the user in a web application. We will see how to use the validation controls in a webform to perform the validations against the data entered by the user. Validation controls are controls which help us to perform different validations on the data entered in controls like Textbox, Listbox and radiobutton.

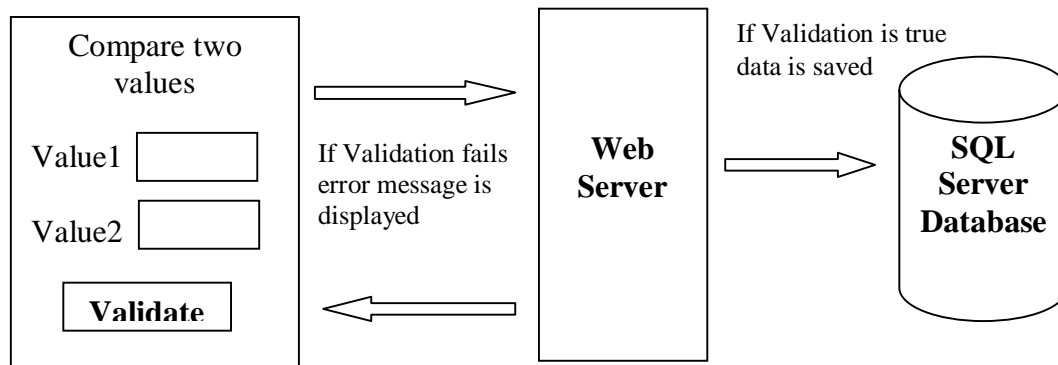


Fig : Working of Validation controls in ASP.NET Page

Using validation controls, we can perform the following validations. We need to check whether user has entered data in a field because the value which is entered into this field is going to be stored in a field which cannot be NULL. Even though, we can check this validation using a stored procedure on the Database server side, it is always better to use validation controls to avoid round trips. Common entries are email ids, telephone numbers, URLs and credit card numbers.

We can add validation controls to a webform just like any other controls. For each specific type of validation, there is a specific validation control available in the .Net framework. The common validation controls we use in a webform are:

- RequiredFieldValidator Control - For compulsory required fields
- RegularExpressionValidator Control - For Validating expressions
- CompareValidator Control- For Comparing values
- RangeValidator control- For checking for a range of values
- ValidationSummary control- For Summarizing Validation Errors

1) Example of Compare Validator Control

```

<html>
<body>

<form runat="server">
<table border="0" bgcolor="#b0c4de">
  <tr valign="top">
    <td colspan="4"><h4>Compare two values</h4></td>
  </tr>
  <tr valign="top">
    <td><asp:TextBox id="txt1" runat="server" /></td>
    <td> = </td>
    <td><asp:TextBox id="txt2" runat="server" /></td>
    <td><asp:Button Text="Validate" runat="server" /></td>
  </tr>
</table>
<br />
<asp:CompareValidator
id="compval"
Display="dynamic"
ControlToValidate="txt1"
ControlToCompare="txt2"
ForeColor="red"
BackColor="yellow"
Type="String"
  >
  </asp:CompareValidator>

```

```

EnableClientScript="false"
Text="Validation Failed!"
runat="server" />
</form>

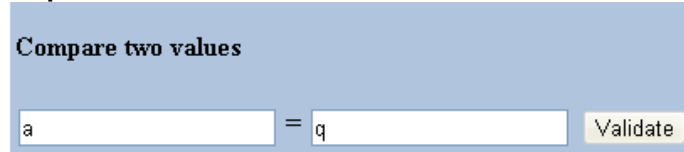
```

```

</body>
</html>

```

Output Result :



Validation Failed!

2) Example of Range Validator control

```

<html>
<body>

<form runat="server">
Enter a date between 2005-01-01 and 2005-12-31:
<br />
<asp:TextBox id="tbox1" runat="server" />
<br /><br />
<asp:Button Text="Submit" runat="server" />
<br /><br />
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="2005-01-01"
MaximumValue="2005-12-31"
Type="Date"
EnableClientScript="false"
Text="The date must be between 2005-01-01 and 2005-12-31!"
runat="server" />
</form>

</body>
</html>

```

Output Result :

Enter a date between 2005-01-01 and 2005-12-31:

The date must be between 2005-01-01 and 2005-12-31!

3.7 USER CONTROL

Server controls are one of the things that make developing with ASP.NET so simple and powerful at the same time. We've discussed HTML and web server controls and have showed you how to use them in your ASP.NET pages, but what if there's not a control that does exactly what you want to do?

Like most everything in ASP.NET there's really no magic involved with server controls. In fact, you can build your own controls and use them on your pages just like you use the ones that ship with .NET. Controls that you build are called user controls and they are shown below.

Writing Your First User Control

Actually, writing a basic user control (also sometimes called pagelets) couldn't be much easier... most any ASP.NET page can be used as a server control. Here's a simple example:

basic.ascx
<pre><p> This is a user control... really! </p></pre>

The customary extension given to pages meant to be used as a control. It's not really anything special, but it does keep things easier to understand and by default .ascx files are prevented from being directly executed and served by IIS. Now that we've created our user control, here's a sample of how to use it in an ASP.NET web page.

basic.aspx
<pre><%@ Page Language="C#" %> <%@ Register TagPrefix="asp101samps" TagName="SomeText" Src="basic.ascx" %> <html> <head> <title>ASP.NET User Control Sample - Basic</title> </head> <body bgcolor="#FFFFFF"> <asp101samps:SomeText runat="server" /> </body> </html></pre>

The above page will output a standard HTML page with the text contained in our user control in place of the user control's tag. How does it do it? The magic here is in the Register directive. To use it you'll need to specify three attributes:

TagPrefix : The TagPrefix attribute defines a namespace in which the control will live. You can think of this as the world in which the control will live. Because of this, you can have different controls with the same name and as long as they have a different TagPrefix (and as a result different namespaces) they can peacefully co-exist on the same page.

TagName : The TagName attribute determines the name by which the control will be referred to. It needs to be unique within the namespace, but you can choose whatever you like. Generally names are chosen that are indicative of what the control actually does.

Src : The Src (or source) attribute simply specifies where the code that makes up the control being defined is to be found. Virtual paths are used so the value should be something like "control.ascx" or "/path/control.ascx" and not a physical path like "C:\path\control.ascx."

Once you've added the Register directive, the control is registered and can be used just like any other server control. You specify the TagPrefix and TagName in the control's tag just like you would with a built in control and make sure you've got the **runat="server"** attribute and you're ready to roll. Here's the basic form for a user control's tag:

```
<TagPrefix:TagName runat="server" />
```

Giving Your Control Properties

This time around I've added two properties to the control - one to control the color and one to control the text.

properties.ascx
<pre><script language="VB" runat="server"> Public Color As String = "black"</pre>

```

        Public Text As String = "This is a user control... really!"
    </script>

    <p>
    <font color="<%= Color %>">
    <%= Text %>
    </font>
    </p>

```

By default the control will look the same as before, but now we can modify the color or text it emits either by setting properties in the control's tag or programmatically from our code (as long as we give the control an id attribute by which to reference it). Also note that I'm able to reuse the same control a number of times in the page and set the properties of each instance independently.

properties.aspx
<pre> <%@ Page Language="C#" %> <%@ Register TagPrefix="asp101samps" TagName="SomeText" Src="properties.ascx" %> <script language="VB" runat="server"> Sub Page_Load(Sender As Object, E As EventArgs) UserCtrl1.Color = "green" UserCtrl1.Text = "This control's properties were " _ & "set programmatically!" End Sub </script> <html> <head> <title>ASP.NET User Control Sample - Properties</title> </head> <body bgcolor="#FFFFFF"> <asp101samps:SomeText runat="server" /> <asp101samps:SomeText Color="red" runat="server" /> <asp101samps:SomeText Text="This is quite cool!" runat="server" /> <asp101samps:SomeText Color="blue" Text="Ain't It?" runat="server" /> <asp101samps:SomeText id="UserCtrl1" runat="server" /> </body> </html> </pre>

Having Your Control Handle Events

You can get your control to do almost anything you want it to. In the next set of code listings the control will handle the Page_Load event all by itself. This type of event handling allows you to write controls that need very little if any maintaining-code to be written in the page hosting the control. The controls can handle all their own events.

In addition, this control encapsulates an asp textbox control.

events.ascx
<pre> <script language="VB" runat="server"> Sub Page_Load(Src As Object, E As EventArgs) Dim strInitialText As String = "Please Enter a Name!" If Page.IsPostBack Then If txtName.Text = strInitialText </pre>

```

        txtName.Text = ""
    End If
Else
    txtName.Text = strInitialText
End If
End Sub

Public Property Name As String
    Get
        Return txtName.Text
    End Get
    Set
        txtName.Text = Value
    End Set
End Property
</script>

Name: <asp:textbox id="txtName" runat="server" />

<asp:RequiredFieldValidator ControlToValidate="txtName"
    id="valtxtName" Display="Dynamic" runat="server">
    Please Enter a Name!
</asp:RequiredFieldValidator>

```

events.aspx

```

<%@ Page Language="VB" ClientTarget="downlevel" %>
<%@ Register TagPrefix="asp101samps" TagName="SomeText"
    Src="properties.ascx" %>
<%@ Register TagPrefix="asp101samps" TagName="TextBox"
    Src="events.ascx" %>

<script language="VB" runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)
        txtLabel.Text = ""

        ' The textbox control handles its own stuff
        ' in its own Page_Load event handler.
    End Sub

    Sub btnSubmit_Click(Sender As Object, E As EventArgs)
        ' Sets the label to the textbox's text
        txtLabel.Text = txtName.Name

        ' I don't need to worry about validation since
        ' my user control does it for me.
    End Sub
</script>

<html>
<head>
<title>ASP.NET User Control Sample - Validation & Events</title>
</head>

<body bgcolor="#FFFFFF">

<form runat="server">

    <asp101samps:TextBox id="txtName" runat="server" />

    <br />

    <asp:button id="btnSubmit" onClick="btnSubmit_Click"

```

```

        text="Submit" runat="server" />

</form>

<asp101samps:SomeText id="txtLabel" runat="server" />

</body>
</html>

```

Check your progress 3.7

Answer the following questions in 1-2 sentences.

a) What is server control?

.....

.....

b) Write any three server controls.

.....

.....

c) What is validation control and user control?

.....

.....

3.8 DATA BINDING CONTROLS

Data-bound Web server controls are controls that can be bound to a data source control to make it easy to display and modify data in your Web application. Data-bound Web server controls are composite controls that combine other ASP.NET Web controls, such as Label and TextBox controls, into a single layout.

The following controls are list controls which support data binding:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

The selectable items in each of the above controls are usually defined by one or more asp:ListItem controls, like this:

```

<html>
<body>
<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>
</body>
</html>

```

However, with data binding we may use a separate source, like a database, an XML file, or a script to fill the list with selectable items. By using an imported source, the data is separated from the HTML, and any changes to the items are made in the separate data source.

Example of ArrayList RadioButtonList

```

<script language = "VB" runat="server">
Sub Page_Load (S as object, e As Event Args)
if Not Page.IsPostBack then
    dim mycountries=New ArrayList
    mycountries.Add("Norway")
    mycountries.Add("Sweden")
    mycountries.Add("France")
    mycountries.Add("Italy")
    mycountries.TrimToSize()
    mycountries.Sort()
    rb.DataSource=mycountries
    rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>

</body>
</html>

```

Output Result :

- ☒ France
- ☐ Italy
- ☐ Norway
- ☐ Sweden

Your favorite country is: France

3.9 CONFIGURATION

The ASP.NET configuration system features an extensible infrastructure that enables you to define configuration settings at the time your ASP.NET applications are first deployed so that you can add or revise configuration settings at any time with minimal impact on operational Web applications and servers.

The ASP.NET configuration system provides the following benefits:

- Configuration information is stored in XML-based text files. Any standard text editor or XML parser can be used to create and edit ASP.NET configuration files.
- ASP.NET configuration settings for the entire Web server is defined in the root configuration file named [C:\WINNT\Microsoft.NET\Framework\version\CONFIG\Machine.config](#).
- Multiple configuration files, all named [Web.config](#), can appear in multiple directories on an ASP.NET Web application server.
- Each [Web.config](#) file applies configuration settings to its own directory and all child directories below it.
- Configuration files in child directories can supply configuration information in addition to that inherited from parent directories, and the child directory

configuration settings can override or modify settings defined in parent directories.

- At run time, ASP.NET uses the configuration information provided by the [Machine.config](#) and all the [Web.config](#) files in a hierarchical directory structure to compute a collection of configuration settings for each unique URL resource. The resulting configuration settings are then cached for all subsequent requests to a resource.
- ASP.NET detects changes to configuration files and automatically applies new configuration settings to Web resources affected by the changes. The server does not have to be rebooted for the changes to take effect. Hierarchical configuration settings are automatically recalculated and recached whenever a configuration file in the hierarchy is changed.
- The ASP.NET configuration system is extensible. You can define new configuration parameters and write configuration section handlers to process them.
- ASP.NET protects configuration files from outside access by configuring Internet Information Services (IIS) to prevent direct browser access to configuration files. HTTP access error 403 (forbidden) is returned to any browser attempting to directly request a configuration file.

3.10 PERSONALIZATION

ASP.NET 2.0 provides specific support for personalizing Web sites. The support ASP.NET provides for personalization service greatly simplifies the whole management and storage of personal information. Defining a Web site's personalization facilities begins with defining User Profiles.

3.10.1 User Profiles

The heart of the new ASP.NET personalization service is the User Profile. A User Profile defines what kind of personal information your Web site needs. For example, you may want to know personal data about users of your Web site, such as name, gender, number of visits to the site, and so forth. User Profiles are also handy for storing user preferences for your site. For example, you might include a Theme as part of a personal profile so that users can tailor the pages to their particular tastes.

Once the personalization properties are defined in Web.Config, a component within .NET has to be able to read it and use it. That job is handled by ASP.NET personalization providers.

3.10.2 Personalization Providers

Providers hide the coding differences involved in creating the necessary objects for connecting to various databases. Just pick a provider (for example, SQL Server or Access), and the provider does the dirty work of manufacturing connections and such. ASP.NET includes two personalization providers out of the box: a profile provider for custom user data and a personalization provider for Web Parts.

ASP.NET defines the fundamental provider capabilities in an abstract class named `PersonalizationProvider`. Those capabilities include such things as loading and saving personalization properties and managing their relationship to any Web Parts used within a site. ASP.NET provides a default implementation of these capabilities in a concrete class named `SqlPersonalizationProvider`, which is derived from `PersonalizationProvider`.

3.10.3 Using Personalization

Using personalization is pretty straightforward. You basically define personalization properties in Web.Config. ASP.NET will synthesize a class you may use to manage personalization settings. At that point, profile information is available in much the same way as session state is available.

3.10.4 Defining Profiles in Web.Config

Profile schema is defined within Web.Config as name/type pairs. Imagine that in the course of designing your site, you decided you'd like to track the following information about a particular user:

- User name
- Gender
- Visit count
- Birthday

Defining these properties is a matter of populating them in Web.Config. A definition for the properties listed above might look like this Web.Config:

```
<system.web>
  <profile automaticSaveEnabled="true" >
    <properties>
      <add name="NumVisits" type="System.Int32"/>
      <add name="UserName" type="System.String"/>
      <add name="Gender" type="bool">
      <add name="Birthday" type="System.DateTime">
    </properties>
  </profile>
</system.web>
```

The personalization properties consist of name/type pairs and will basically become the schema under which the personalization data will be stored. Once defined in the Web.Config file, the profile may be used in the site through the Profile property found in the current HttpContext (and is also available via the Page).

3.10.5 Use Profile Information

To use the profile in the Web site, you access it in much the same way you might access session state. However, instead of being represented by name/value pairs accessed through an indexer, the ASP.NET compiler will synthesize a profile object based upon the scheme defined in the Web.Config file.

For example, given the schema listed above, ASP.NET will synthesize a class named ProfileCommon, based upon the ProfileBase class. The synthesized class will reflect the instructions written into the Web.Config by inserting properties, shown here in bold:

```
public class ProfileCommon : ProfileBase
{
  public virtual HttpProfile GetProfile(string username);
  public object GetPropertyValue(string propertyName);
  public void SetPropertyValue(string propertyName,
    object propertyValue);
  public HttpProfileGroupBase GetProfileGroup(String groupName);
  public void Initialize(String username, Boolean isAuthenticated);
  public virtual void Save();
  public void Initialize(SettingsContext context,
    SettingsPropertyCollection properties,
    SettingsProviderCollection providers);
  public string UserName{get; set;};
  public int NumVisits{get; set;};
  public bool Gender{get; set; };
  public DateTime Birthdate{get; set; };
}
```

To access the profile properties, simply use the Profile property within the page. The Profile property is an instance of the Profile Common class synthesized by ASP.NET. Just access the members of the Profile, like so:

```
protected void Page_Load(object sender, EventArgs e)
{
  if (Profile.Name != null)
  {
```

```

        Response.Write("Hello " + Profile.Name);
        Response.Write("Your birthday is " +
            Profile.Birthdate);
    }
}

```

3.10.6 Saving Profile Changes

The preceding code snippet assumes there's already personalization information associated with the user. To insert profile data for a particular user, simply set the properties of the Profile object. For example, imagine a page that includes a handler for saving the profile. It might look something like this:

```

protected void ProfileSaveClicked(object sender, EventArgs e)
{
    Profile.Name = this.TextBoxName.Text;
    Profile.Birthdate = this.Calendar1.SelectedDate;
}

```

The easiest way to ensure that the personalization properties persist is to set the `AutomaticSaveEnabled` to `true`. Personal profile data will be saved automatically by the provider. Alternatively, you may call `Profile.Save` as necessary to save the personalization properties. In addition to saving and loading profiles, you may also delete the profile for a specific user by calling `Profile.DeleteProfile`.

3.10.7 Profiles and Users

Profile information is associated with the current user based upon the identity of the user. By default, ASP.NET uses the `User.Identity.Name` within the current `HttpContext` as the key to store data. By default, profiles are available only for authenticated users.

ASP.NET supports anonymous profiles as well. Turn this on within `Web.Config`. The default tracking mechanism for anonymous profiles is to use cookies. However, as with tracking session state, you may tell ASP.NET to use a mangled URL.

The following exercise illustrates using personalization profiles based on the user's login ID.

Using Profiles

1. Create a new project. Name the project `MakeItPersonal`.
2. Add a `Web.Config` file to the project. Update `Web.Config` to include some profile properties. The example here includes a user name, a Theme, and a birth date. Be sure to turn anonymous Identification to true. The following example shows that you may group and nest profile structures using the `<group>` element.

`<system.web>`

```

<profile>
  <properties >
    <add name="Theme" type="System.String"/>
    <add name="Name" type="String"/>
    <add name="Birthdate" type="System.DateTime"/>
    <group name="Address">
      <add name="StreetAddress"/>
      <add name="City"/>
      <add name="State"/>
      <add name="ZipCode"/>
    </group>
  </properties>
</profile>

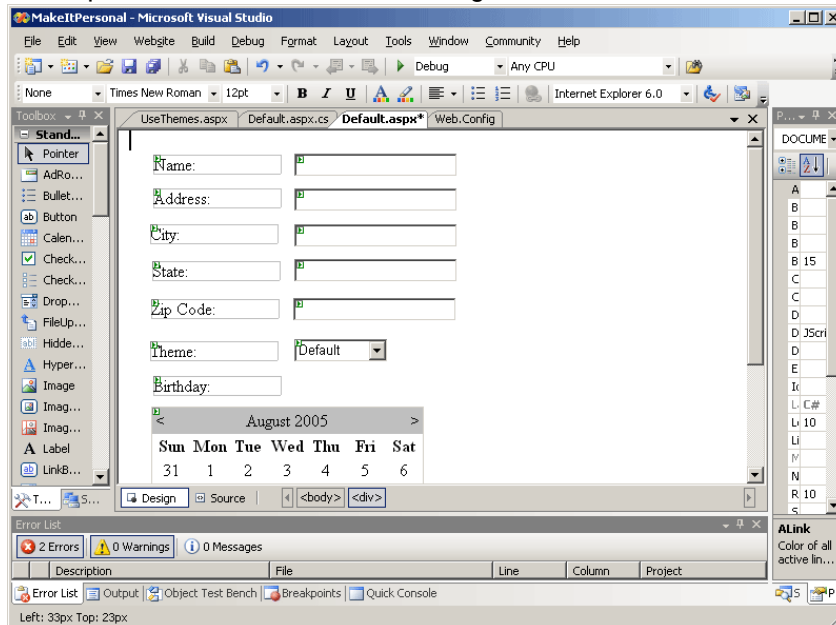
```

`</system.web>`

1. Borrow the Default and SeeingRed Themes from the `MasterPagesSite` project. This will let the user pick the Theme.
2. Borrow the `UseThemes.aspx` and `.cs` files from the `MasterPagesSite` project.
3. Borrow the `Banner.ascx` file from the `MasterPagesSite`.

- Now update the Default.aspx page. This will be where users type profile information. Add text boxes for the name, address, city, state, and zip code. Add a drop-down list box populated with Default and SeeingRed items. This will be used for selecting the Theme. Also add a calendar control to pick the birthdate.
- Add a button the user may click to submit profile information. Add a handler to input these values into the profile. Double-click on the button to add the handler.

The input screen should look something like this:



- Update Page_Load to display profile information (if it's there). Grab the profile object and set each of the text boxes and the calendar control.

```
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
```

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            ProfileCommon pc = this.Profile.GetProfile(Profile.UserName);
            if (pc != null)
            {
                this.TextBoxName.Text = pc.Name;
                this.TextBoxAddress.Text = pc.Address.StreetAddress;
                this.TextBoxCity.Text = pc.Address.City;
                this.TextBoxState.Text = pc.Address.State;
                this.TextBoxZipCode.Text = pc.Address.ZipCode;
                this.DropDownList1.SelectedValue = pc.Theme;
                this.Calendar1.SelectedDate = pc.Birthdate;
            }
        }
    }
    // ...
}
```

- Update the profile submission handler to store the profile information.

```
protected void ButtonSubmitProfile_Click(object sender, EventArgs e)
```

```

{
    ProfileCommon pc = this.Profile.GetProfile(Profile.UserName);

    if (pc != null)
    {
        pc.Name = this.TextBoxName.Text;
        pc.Address.StreetAddress = this.TextBoxAddress.Text;
        pc.Address.City = this.TextBoxCity.Text;
        pc.Address.State = this.TextBoxState.Text;
        pc.Address.ZipCode = this.TextBoxZipCode.Text;
        pc.Theme = this.DropDownList1.SelectedValue;
        pc.Birthdate = this.Calendar1.SelectedDate;

        pc.Save();
    }
}

```

8. Finally, update the UseThemes.aspx page to use the Theme. Override the page's OnPreInit method. Have the code apply the Theme as specified by the profile.

```

protected override void OnPreInit(EventArgs e)
{
    ProfileCommon pc = this.Profile.GetProfile(Profile.UserName);
    if (pc != null)
    {
        String strTheme = pc.Theme.ToString();
        if (strTheme != null &&
            strTheme.Length > 0)
        {
            this.Theme = strTheme;
        }
    }
    base.OnPreInit(e)
}

```

9. When you surf to the page, you should be able to enter the profile information and submit it. Following your initial visit, the profile will be available whenever you hit the site.

3.11 SESSION STATE

ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application.

What is session state?

A session is defined as the period of time that a unique user interacts with a Web application. Session state is a collection of objects, tied to a session are stored on a server.

A session is defined as the period of time that a unique user interacts with a Web application. Active Server Pages (ASP) developers who wish to retain data for unique user sessions can use an intrinsic feature known as session state.

Programmatically, session state is nothing more than memory in the shape of a dictionary or hash table, e.g. key-value pairs, which can be set and read for the duration of a user's session. For example, a user selects stocks to track and the Web application can store these values in the user's ASP session instance:

```
Session("Stocks") = "MSFT; VRSN; GE"
```

On subsequent pages these values are read and the Web application has access to these values without the user re-entering them:

```
' Get Stocks, split string, etc.
String StockString
```

```
StockString = Session("Stocks")
```

ASP maintains session state by providing the client with a unique key assigned to the user when the session begins. This key is stored in an HTTP cookie that the client sends to the server on each request. The server can then read the key from the cookie and re-inflate the server session state.

ASP.NET Session State

- **Process independent.** ASP.NET session state is able to run in a separate process from the ASP.NET host process. If session state is in a separate process, the ASP.NET process can come and go while the session state process remains available. Of course, you can still use session state in process similar to classic ASP, too.
- **Support for server farm configurations.** By moving to an out-of-process model, ASP.NET also solves the server farm problem. The new out-of-process model allows all servers in the farm to share a session state process. You can implement this by changing the ASP.NET configuration to point to a common server.
- **Cookie independent.** Although solutions to the problem of cookie less state management do exist for classic ASP, they're not trivial to implement. ASP.NET, on the other hand, reduces the complexities of cookie less session state to a simple configuration setting.

Let's look at each of these features in more detail, including how the settings are configured.

Using ASP.NET Session State

Session state settings in ASP.NET are configured through the ASP.NET XML configuration file **config.web**. We'll look at **config.web** in more detail in a later column, but for this discussion of session state let's look at it briefly.

Config.web

There are two types of configuration files: a machine configuration file and an application configuration file, both named **config.web**. The two are identical, except that the machine configuration file applies settings to all applications but the application configuration files are either restrictive or expansive on an application-by-application basis.

The machine **config.web** file is in the **WinNT\Microsoft.NET\Framework\v1.0.2204** directory, while the optional application configuration files exist in the application's directory. Application **config.web** files are optional in the sense that if an application **config.web** file doesn't exist, the machine **config.web** settings are used instead. ASP.NET session state settings can be made in the machine **config.web** file and overridden in a particular application's **config.web** file.

Session configuration

Below is a sample **config.web** file used to configure the session state settings for an ASP.NET application:

```
<configuration>
  <sessionstate
    mode="inproc"
    cookieless="false"
    timeout="20"
    sqlconnectionstring="data source=127.0.0.1;user id=<user
id>;password=<password>"
    server="127.0.0.1"
    port="42424"
  />
</configuration>
```

The settings above are used to configure ASP.NET session state. Let's look at each in more detail and cover the various uses afterward.

- **Mode.** The mode setting supports three options: inproc, sqlserver, and stateserver. As stated earlier, ASP.NET supports two modes: in process and out of process. There are also two options for out-of-process state management: memory based (stateserver), and SQL Server based (sqlserver). We'll discuss implementing these options shortly.
- **Cookieless.** The cookieless option for ASP.NET is configured with this simple Boolean setting.
- **Timeout.** This option controls the length of time a session is considered valid. The session timeout is a sliding value; on each request the timeout period is set to the current time plus the timeout value
- **Sqlconnectionstring.** The sqlconnectionstring identifies the database connection string that names the database used for mode sqlserver.
- **Server.** In the out-of-process mode stateserver, it names the server that is running the required Windows NT service: ASPState.
- **Port.** The port setting, which accompanies the server setting, identifies the port number that corresponds to the server setting for mode stateserver.

Sample session state application

Before we use session state, we need an application to test it with. Below is the code for a simple Visual Basic application that writes to and reads from session state, **SessionState.aspx**:

```
<Script language = "VB" runat="server">
Sub Session_Add(sender As Object, e As EventArgs)
    Session("MySession") = text1.Value
    span1.InnerHtml = "Session data updated! <P>
        Your session contains: <font color=red>" +
        Session("MySession").ToString() + "</font>"
End Sub

Sub CheckSession(sender As Object, e As EventArgs)
    If (Session("MySession") = Isnull) Then
        span1.InnerHtml = "NOTHING, SESSION DATA LOST!"
    Else
        span1.InnerHtml = "Your session contains:
            <font color=red>" +
            Session("MySession").ToString() + "</font>"
    End If
End Sub
</Script>

<form runat=server>
<input id=text1 type=text runat=server>
<input type=submit runat=server
    OnServerClick="Session_Add" Value="Add to Session State">
<input type=submit runat=server
    OnServerClick="CheckSession" Value="View Session State">
</form>
<hr size=1>
<font size=6><span id=span1 runat=server/></font>
```

This simple page wires up two server-side events for the **Add** and **View** buttons, and simply sets the session state to the value in the text box.

3.12 SUMMARY

- ASP.NET is the latest version of Microsoft's Active Server Pages technology (ASP). ASP.NET is a server side scripting technology that enables scripts to be executed by an Internet server.

- A master page is a template for other pages, with shared layout and functionality. The master page defines placeholders for content pages.
- Web Forms Page Life Cycle for a Web Forms page is similar to that of any Web process that runs on the server. Certain characteristics of Web processing information is passed through HTTP protocol, the stateless nature of Web pages, the ASP.NET page framework performs many Web application services for us. It is important to understand the sequence of events that occur when a Web Forms page is processed.
- ASP.NET server controls are objects and have their own properties, methods and event handlers. HTML elements in ASP.NET files are, by default, treated as text. To make these elements programmable, add a `runat="server"` attribute to the HTML element. This attribute indicates that the element should be treated as a server control.
- The support ASP.NET provides for personalization service greatly simplifies the whole management and storage of personal information. Defining a Web site's personalization facilities begins with defining User Profiles.
- A session is defined as the period of time that a unique user interacts with a Web application. Session state is a collection of objects, tied to a session are stored on a server.

Check your progress 3.11

Answer the following questions in 1-2 sentence\cues.

a) What is ASP.net configuration?

.....

b) What is personalization in ASP.net?

.....

c) What is session state?

.....

3.13 CHECK YOUR PROGRESS -ANSWERS

3.2

1. It is a server side scripting language, means complete code get run on server and result get back in HTML format to client.
2. Master page, themes, role, personalization, new ado.net features, XML standard support etc.

3.3

1. Web form life cycle refers to the tenure of page in server from initiating to out it consist of : -
 Page request → Start → Page initialization → Load → Validation → Post back event handling → Rendering → Unload.

3.4 - 3.7

1. Server Control runs on server side, they have their own property, they retain their value during subsequent request.
2. Validation control are for defined control which validate the userdata entering in a webpage. Ex. Required field Validator user control is a type of composite control which works like other ASP Control, but the basic difference is that

control are designed by user itself. Ex. Combination of textbox and command button.

3.8 – 3.11

1. It is the configuration defined at the time we host our WebPages and it is stored in web.config file.
2. Personalization is a process where we personalize a web page according to user, which include language, common page etc.
3. Session state is a time define that how long a user connected to a page or could be able to connect that page. Ex. Your email session expires in 5 min ideal time.

3.14 QUESTIONS FOR SELF - STUDY

1. Explain Life Cycle of a Web Forms Page with diagram.
2. Explain Dropdown List Server Control with example.
3. What is Validation controls explain with example?
4. Explain the benefits of configuration.

3.15 SUGGESTED READINGS

1. Programming C#: Building .NET Applications with C# By Jesse Liberty
2. Learning C# 2005 by Jesse Liberty
3. Microsoft MSDN
4. <http://netservicesindia.com/blog/?p=35>
5. <https://en.wikipedia.org/wiki/ASP.NET>
6. http://www.zeali.net/mirrors/w3cshool/aspnet/aspnet_newfeatures.asp.html
7. http://www.w3schools.com/aspnet/aspnet_arraylist.asp



NOTES

[illegible]

NOTES

[illegible]

Declaring Variables in ASP.NET

4.0	Objectives
4.1	Introduction
4.2	Assignment and Initialization of variable
4.3	Arrays
4.4	Enumerations
4.5	Variable Operations
4.5.1	Advance & Math Operations
4.5.2	Type Conversions
4.6	Object Based Manipulation
4.6.1	The string object
4.6.2	Date Time object and Time span types object
4.6.3	The Array Object
4.7	Conditional Structures
4.8	Loop Structures
4.9	Functions & Subroutines
4.10	Summary
4.11	Check your Progress – Answers
4.12	Questions for Self – Study
4.13	Suggested Readings

4.0 OBJECTIVES

After studying this chapter you will be able to –

- ♦ explain various data types
- ♦ explain Arrays and Enumerations
- ♦ discuss various conditional structures and examples.
- ♦ describe the variable operations
- ♦ explain Function and its use.

4.1 INTRODUCTION

As with all programming languages, you keep track of data in C# using *variables*. Variables can store numbers, text, dates, and times, and they can even point to full-fledged objects.

When you declare a variable, you give it a name, and you specify the type of data it will store. To declare a local variable, you start the line with the data type, followed by the name you want to use. A final semicolon ends the statement:

```
// Declare an integer variable named errorCode  
int errorCode ;
```

```
// Declare a string variable named myName
string myName ;
```

Every .NET languages use the same variable data types. Different languages may provide slightly different names (for example, a VB Integer is the same as in C# int), but the CLR make no distinction, in fact, they are just two different names for the same base data type. This design allows for deep language integration. Because languages share the same core data types, you can easily use objects written in one .NET language in an application written in another .Net language. No data type conversions are required.

To create this common data type system, Microsoft needed to iron out many of the inconsistencies that existed between VBScript, VB6, C++, and other languages. The solution was to create a set of basic data types, which are provided in the .NET Class library.

Table 4-1 lists the most important core data types.

C# Name	VB Name	.NET Type Name	Contains
byte	Byte	Byte	An integer from 0 to 255.
short	Short	Int16	An integer from -32,768 to 32,767
int	Integer	Int32	An integer from -2,147,483,648 to 2,147,483,647.
long	Long	Int64	An integer from about -9.2e18 to 9.2e18.
float	Single	Single	A single-precision floating point number from approximately -3.4e38 to 3.4e38 (for big numbers) or -1.5e-45 to 1.5e-45 (for small fractional numbers).
double	Double	Double	A double-precision floating point number from approximately -1.8e308 to 1.8e308 (for big numbers) or -5.0e-324 to 5.0e-324 (for small fractional numbers).
decimal	Decimal	Decimal	A 128-bit fixed-point fractional number that supports up to 28 significant digits.
char	Char	Char	A single 16-bit Unicode character.
string	String	String	A variable-length series of Unicode characters.
bool	Boolean	Boolean	A true or false value.
*	Date	DateTime	Represents any date and time from 12.00.00 AM, January 1 of the year 1 in the Gregorian calendar, to 11.59.59 PM, December 31 of the year 9999. Times values can resolve values to 100 nano-second increments. Internally, this data type is stored as a 64-bit integer.
*	*	Timespan	Represents a period of time, as in ten seconds or three days. The smallest possible interval is 1 <i>tick</i> (100 nanoseconds).
object	Object	Object	The ultimate base class of all .NET types. Can contain any data type or object.

You can also declare a variable by using the type name from the .NET class library. This approach produces identical variables. It's also a requirement when the data type doesn't have an alias built into the language. For example, you can rewrite the earlier example that used C# data type name with this code snippet that uses the class library names:

```
System.Int32 errorCode ;
System.String myName ;
```

This code snippet uses fully qualified type names that indicate that the Int32 type and the String type are found in the System namespace (along with all the most fundamental types). In Chapter 3, you'll learn about types and namespaces in more detail.

4.2 ASSIGNMENT AND INITIALIZATION OF VARIABLE

Once you've declared your variable, you can freely assign values to them, as long as these values have the correct data type.

Here's the code that shows this two-steps process:

```
// Declare variables.
int errorCode ;
string myName ;

//Assign Values.
errorCode = 10 ;
myName = "Matthew";
```

You can also assign a value to a variable in the same line that you declare it. This example compresses four lines of code in two:

```
int errorCode = 10 ;
string myName = "Matthew";
```

C# safeguards you from errors by restricting you from uninitialized variables. This means the following code will not succeed:

```
int number ;           // Number is uninitialized
number = number + 1 ;   // This will cause a compiler error
The proper way to write this code is to explicitly initialize the number variable to an
appropriate value, such as 0, before using it:
int number = 0 ;        // Number now contains 0.
number = number + 1 ;    // Number now contains 1.
```

C# also deals strictly with data types. For example, the following code statement won't work as written:

```
decimal myDecimal = 14.5 ;
```

The problem is that the literal 14.5 automatically interpreted as a double, and you can't convert a double to a decimal without using casting syntax, which is described later in this chapter. To get around this problem, C# defines a few special characters that you can append to literal values to indicate their data type so that no conversion will be required.

These are as follow:

M (decimal)
D (double)
F (float)
L (long)

For example, you can rewrite the earlier example using the decimal indicator as follow:
decimal myDecimal = 14.5M ;

Interestingly, if you're using code like this to declare and initialize variable in one step, and if the C# compiler can determine the right data type based on the value you're using, you don't need to specify the data type. Instead, you can use the all-purpose *var* keyword in place of the data type.

That means the previous line of code is equivalent to this:

```
var myDecimal = 14.5M ;
```

Here the compiler realizes that a decimal data type is the most appropriate choice for the myDecimal variable, and uses that data type automatically. There is no performance difference. The myDecimal variable that you create using an inferred data type behaves in exactly the same way as a myDecimal variable created with an explicit data type. In fact, the low-level code that the compiler generates is identical. The only difference is that the *var* keyword saves some typing.

Many C# programmers feel uneasy with the *var* keyword because it makes code less clear. However, the *var* keyword is a more useful shortcut when creating objects, as you'll see in the next chapter.

Strings and Escape Characters

C# treats text a little different than other languages such as VB. It interprets any embedded backslash (\) as the start of a special character escape sequence. For example, \n means add a new line (carriage return).

The most useful character literals are as follows:

```
" (double quote)
\n (new line)
\t (horizontal tab)
\\ (backward slash)
```

You can also insert a special character based on its hex code using the syntax \x123. This inserts a single character with hex value 123

Note that in order to specify the actual backslash character (for example, in a directory name), you require two slashes.

Example:

```
// A C# variable holding the
// C:\MyApp\MyFiles path.
path = "C:\\MyApp\\MyFiles" ;
Alternatively, you can turn off C# escaping by preceding a string with @ symbol, as
shown here:
path = @"C:\MyApp\MyFiles" ;
```

Check your progress 4.2

Answer the following questions in 1-2 sentences.

a) State the various data types.

.....

.....

b) What is a string and escaped characters?

.....

.....

4.3 ARRAYS

Arrays allow you to store a series of values that have the same data type. Each individual value in the array is accessed using one or more index numbers. It's often convenient to picture arrays as list of data (if the array has more than one dimension) or grids of data (if the array has two dimensions). Typically, arrays are laid out contiguously in memory.

All arrays start at a fixed lower bound. This rule has no exceptions. When you create an array in C#, you specify the numbers of elements. Because counting starts at 0, the highest index is actually one fewer than the number of elements.

```
// create an array with four strings (from index 0 to index 3).
```

```
// You need to initialize the array with the new keyword in order to use it:
```

```
string [ ] vStringArray1 = new string [4] ;
```

```
// Create a 2x4 grid array (with a total of 8 integers).
```

```
int [ , ] vIntArray1 = new int [2, 4] ;
```

By default, if your array includes simple data types, they are all initialized to default values (0 or false), depending on whether you are using some type of number or a Boolean variable. You can also fill in an array with data at the same time that you create it. In this case, you don't need to explicitly specify the number of elements, because .NET can determine it automatically

```
// Create an array with four strings, one for each number from 1 to 4.
```

```
string [ ] vStringArray2 = { "1" , "2" , "3" , "4" } ;
```

The same technique works for multidimensional arrays, except that two sets of curly brackets are required:

```
// Create a 4x2 array (a grid with four rows and two columns).  
int [ , ] vIntArray2 = {{1, 2},{3, 4},{5, 6},{7, 8}} ;
```

Figure 3-1 shows what this array looks like in memory:

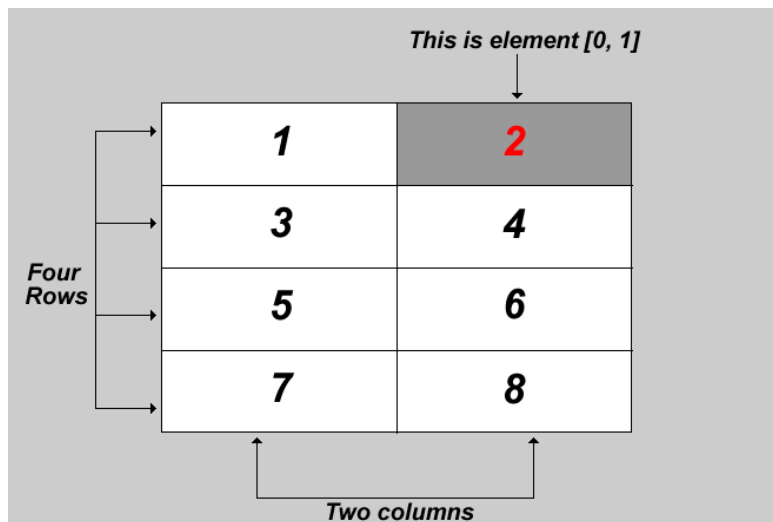


Figure 4.1 A sample array of integers

To access an element in an array, you specify the corresponding index number in square bracket: []. Array indices are always zero-based. That means that `vMyArray[0]` accesses the first cell in a one dimensional array, `vMyArray[2]` accesses the second cell, and so on:

```
// Access the value in row 0 (first row), column 1 (second column).  
int vElement = vIntArray2[0, 1] ;
```

The ArrayList

C# arrays do not support redimensioning. This means that once you create an array, you can't change its size. Instead, you would need to create a new array with the new size and copy values from the old array to the new, which would be a tedious process. However, if you need a dynamic-library array-like list, you can use one of the collection classes provided to all.NET languages through the .NET class library. One of the simplest collection classes that .NET offers is the `ArrayList`, which always allows dynamic resizing.

Here's a snippet of C# code that uses an `ArrayList`:

```
// Create an ArrayList object. It's a collection, not an array,  
// So the syntax is slightly different  
ArrayList vDynamicList = new ArrayList ( ) ;  
  
// Add several strings to the list.  
// The ArrayList is not strongly typed, so you can add any data type  
// although its simplest if you store just one type of object  
// in any given collection.  
vDynamicList.Add( "One" ) ;  
vDynamicList.Add( "Two" ) ;  
vDynamicList.Add( "Three" ) ;  
vDynamicList.Add( "Four" ) ;
```

```
// Retrieve the first string. Notice that the object must be converted to a
// string, because there's no way for .NET to be certain what it is.
string vItem1 = Convert .ToString( vDynamicList[0] ) ;
```

Source : Beginning ASP.NET 4.5 in C# By Matthew MacDonald

4.4 ENUMERATIONS

An enumeration is a group of related constants; each of which is given a descriptive name. Each value in an enumeration corresponds to a preset integer. In your code, however, you can refer to an enumerated value by name, which makes your code clearer and helps prevent errors. For example, it's much more straightforward to set the border of a label to the enumerated value `BorderStyle.Dashed` rather than the obscure numeric constant 3. In this case, `Dashed` is a value in the `BorderStyle` enumeration, and it represents the number 3.

Here's an example of an enumeration that defines different types of users:

```
// Define an enumeration type named UserType with three possible values.
enum UserType
{
    Admin,
    Guest,
    Invalid
}
```

Now you can use the `UserType` enumeration as a special data type that is restricted to one of three possible values. You assign or compare the enumerated value using the dot notation shown in the following example:

```
// Create a new value and set it equal to the UserType.Admin constant.
UserType vNewUserType = UserType .Admin ;
```

Internally, enumerations are maintained as numbers. In the preceding example 0 is automatically assigned to `Admin`, 1 to `Guest`, and 2 to `Invalid`. You can set a number directly in an enumeration variable; although this can lead to undetected error if you use a number that doesn't correspond to one of the defined values. In some scenarios, you might want to control what numbers are used for various values in an enumeration. This technique is typically used when the number has some specific meaning or corresponds to some other piece of information.

For example, the following code defines an enumeration that represents the error code returned by a legacy component:

```
enum ErrorCode
{
    NoResponse = 166 ,
    TooBusy = 167 ,
    Pass = 0
}
```

Now you can use the `ErrorCode` enumeration with a function that returns an integer representing an Error Condition as shown here:

```
ErrorCode vError ;
vError = Dosomething() ;
if (vError == ErrorCode .Pass )
{
    // Operation Succeeded.
}
```

Clearly, enumerations create more readable code. They also simplify coding, because once you type in the enumerations type name (`ErrorCode`) and add the dot (`.`), Visual Studio will pop up a list of possible values using IntelliSense.

Examples :

The following sample will demonstrate arrays. Drag a button and a textbox control on to the Web Forms page. Open the code designer window and paste the following code.

```

Private void Button1_Click ( System.Object Sender, System.EventArgs e) {
String [ ] Sports = new String [6];
Sports[0]="Tennis"
Sports[1] = "Cricket"
Sports[2] = "Rubgy"
Sports[3] = "Aussie Rules"
Sports[4] = "Soccer"
Sports[5] = "Hockey"
Label1.Text = "The sport in the location you entered is" + " " +
Sports(Convert.ToInt32(TextBox1.Text));
}

```

Output Result

Enter a number between 0 and 5 to get the Sport stored in that location

Create Array and get Sport

The sport in the location you entered is Aussie Rules

The above sample code creates a Sports array with six elements, asks the user to enter a number between 0 and 5 and displays the sport that is stored in the location (number) the user entered on a label when the button is clicked.

Strings

Strings are supported by the .NET String class. The String data type can represent a series of characters and can contain approximately up to 2 billion Unicode characters. There are many built-in functions in the String class. Some .NET Framework functions are also built into the String class. Some common String functions are discussed below:

LCase: Converts a string to lower case

UCase: Converts a string to upper case

Len: Calculates the number of characters in the string

LTrim: Omits spaces that appear on the left-side of the string

RTrim: Omits spaces that appear on the right-side of the string

Trim: Omits both leading and trailing spaces

Clone: Clones a string

Concat: Joins two strings

Space: Used to create a string with spaces

Left: Takes two arguments and returns a string that consists of the left-most characters of the string sent

Right: Takes two arguments and returns a string that consists of the right-most characters of the string sent

Instr: Searches and returns a shorter string within a long string

Replace: Searches for a small string in a long string and replaces it with the specified string

Sample Application

The following is a simple and interesting sample. It asks the user to enter some information and generates a username and password based on the information entered and displays the newly generated login information to the user. To start, drag six labels, four textboxes and a button control on to the Web Forms page. TextBox1 accepts first name, TextBox2 accepts last name, TextBox3 accepts age and TextBox4 accepts country. Look at the sample below for the user interface. Open the code designer window and paste the following code.

```

Private void Button1_Click(System.Object Sender, system.EventArgs e)
String mystr, myStr1, mystring, midd;
Mystr = String.left (TextBox1.text,3);
mystr1 = String.right (TextBox2.text,3);
midd =String.mid(textBox4.text, 2, 6) + convert. To int32 (Textbox3.text)-11);
mystring = mystr + mystr1 + textbox3.text;

```

```

Label5.Text = "Your user name is" + " " + mystring;
Label6.Text = "Your random password " + " " + midd;
TextBox1.Visible = False;
TextBox2.Visible = False;
TextBox3.Visible = False;
TextBox4.Visible = False;
Label1.Visible = False;
Label2.Visible = False;
Label3.Visible = False;
Label4.Visible = False;
}

```

Output Result

Strings

First Name	<input type="text" value="Aditya"/>
Last Name	<input type="text" value="Roy"/>
Age	<input type="text" value="24"/>
Enter Country	<input type="text" value="India"/>
<input type="button" value="Submit"/>	

Your user name is AdiRoy24

Your random password generated by our server is ndia13

Check your progress 4.4

Answer the following questions in 1-2 sentences.

a) What is an Array?

.....

b) What is an Enumeration?

.....

4.5 VARIABLE OPERATIONS

You can use all the standards types of variables operations in C#. When working with numbers, you can use various math symbols, as listed in Table. C# follows the conventional order of operations, performing exponentiation first, followed by multiplication and division and then addition and subtraction. You can also control order by grouping sub expressions with parentheses:

```

int vNumber;
vNumber = 4 + 2 * 3 ;
// vNumber will be 10.
vNumber = ( 4 + 2 ) * 3 ;
// vNumber will be 18.

```


Operator	Description	Example
+	Addition	1 + 1 = 2
-	Subtraction	5 - 2 = 3
*	Multiplication	2 * 5 = 10
/	Division	5.0 / 2 = 2.5
%	Gets the remainder left after integer division	7 % 3 = 1 (7 % 3 = (3 + 3 = 6) leaves 1 as result)

Divisions can sometimes cause confusion in C#. If you divide one integer by another integer, C# performs integer division. That means it automatically discards the fractional part of the answer and returns the whole part as an integer. For example, if you divide 5 by 2, you'll end up with 2 instead of 2.5.

The solution is to explicitly indicate that one of your numbers is a fractional value. For example, if you replace 5 with 5M, C# will treat the 5 as a decimal. If you replace 5 with 5.0, C# will treat it as a double. Either way, the division will return the expected value of 2.5. Of course, this problem doesn't occur very often in real-world code, because then you're usually dividing one *variable* by other *variable*.

The operators in Table are designed for manipulating number. However, C# also allows you to use the addition operator (+) to join two strings:

```
// Join three strings together
vMyName = vFirstName + " " + vLastName ;
In addition, C# also provides special assignments operators. Here are a few examples:
```

```
// Add 10 to vMyValue. This is the same as vMyValue = vMyValue + 10 ;
vMyValue += 10 ;
```

```
// Multiply vMyValue by 3. This is the same as vMyValue = vMyValue * 3 ;
vMyValue *= 3 ;
```

```
// Divide vMyValue by 12. vMyValue. This is the same as vMyValue = vMyValue / 10 ;
vMyValue /= 10 ;
```

4.5.1 Advanced Math Operations

In the past, every language has its own set of keywords for common math operations such as rounding and trigonometry. In .NET languages, many of these keywords remain. However, you can also use a centralized Math class that's part of the .NET Framework. This has the pleasant side effect of ensuring that the code you use to perform mathematical operations can easily be translated into equivalent statements in any .NET language with minimal fuss.

To use the math operations, you invoke the methods of the System.Math class. These methods are static, which means they are always available and ready to use. (The next Chapter explores the difference between static and instance members in more detail.)

The following code snippet shows some sample calculations that you can perform with the math class:

```
double vMyValue ;
vMyValue = Math.Sqrt (81) ; // vMyValue = 9.0 (Returns the square root of a
specified number.)
```

```
vMyValue = Math.Round (42.889,2) ; // vMyValue = 42.889 (Rounds a double-
precision floating-point value to the nearest integer.)
```

```
vMyValue = Math.Abs (-10) ; // vMyValue = 10.0 (Returns the absolute value of an 8-
bit signed integer.)
```

```
vMyValue = Math.Log (24.212) ; // vMyValue = 3,18684837802944 (Returns the
natural (base e) logarithm of a specified number. )
```

```
vMyValue = Math.PI ; // vMyValue = 3,14159265358979 (Returns the
number PI.)
```

The features of the Math class are too numerous to list here in their entirety. The preceding examples show some common numeric operations. For more information about the trigonometric and logarithmic functions that are available, refer to the visual Studio Help reference for the Math class.

4.5.2 Type Conversions

Converting information from one data type to another is a fairly common programming task. For example, you might retrieve text input for a user that contains numbers you want to use for a calculation. Or, you might need to take a calculated value and transform it into text you can display in a web page.

Conversions are of two types: widening and narrowing. *Widening* conversions always succeed. For example, you can convert a 32-bit integer into a 64-integer. You won't need any special code.

Here is an example:

```
int vMySmallValue;  
long vMyLargeValue;  
// Get the largest value possible value that can be stored as a 32-bit integer.  
// .NET provides a constant named Int32.MaxValue that provides this number.  
vMySmallValue = Int32.MaxValue ;  
// This always succeeds. No matter how large vMySmallValue is,  
// it can be contained in vMyLargeValue.  
vMyLargeValue = vMySmallValue ;
```

On the other hand, *narrowing* conversions may or may not succeed, depending on the data. If you're converting a 32-bit integer to a 16-bit integer, you could encounter an error if the 32-bit number is larger than the maximum value that can be stored in the 16-bit data type. All narrowing conversions must be performed explicitly. C# uses an elegant method for explicit type conversion. To convert a variable, you simply need to specify the type in parentheses before the expression you're converting.

The following code shows how to change a 32-bit integer to a 16-bit integer:

```
int vCount32 = 1000 ;  
short vCount16;  
  
// Convert the 32-bit integer to a 16-bit integer.  
// If vCount32 is too large to fit, .NET will discard some of the  
// information you need, and the resulting number will be incorrect.  
vCount16 = ( short )vCount32 ;
```

If you don't use an explicit cast when you attempt to perform a narrowing conversion, you'll receive an error when you try to compile your code. However, even if you perform an explicit conversion, you could still end up with a problem.

For example, consider the code shown here, which causes an overflow:

```
int vMySmallValue;  
long vMyLargeValue;  
  
vMyLargeValue = Int64.MaxValue ;  
  
// This will appear to succeed (there won't be an error at runtime),  
// but your data will be incorrect because vMySmallValue cannot  
// hold a value this large.  
vMySmallValue = ( int )vMyLargeValue ;
```

The .Net languages differ in how they handle this problem. In VB, you'll always receive a runtime error that you must intercept and respond to. In C#, however, you'll simply bind with incorrect data in vMySmallValue. To avoid this problem, you should either check that your data is not too large before you attempt a narrowing conversion (which is always a good idea) or use a checked block. The checked block enables overflow checking for a portion of code.

If an overflow occurs, you'll automatically receive an error, just like you would in VB: checked

```
{  
    // This will cause an exception to be thrown.  
    vMySmallValue = ( int )vMyLargeValue ;  
}
```

In C#, you can't use casting to convert numbers to strings, or vice versa. In this case, the data isn't just being moved from one variable to another, it needs to be translated to a completely different format. Thankfully, .NET has a number of solutions for performing advanced conversions. One option is to use the static methods of the Convert class, which support many common data types such as strings, dates, and numbers.

```
string vCountString = "10" ;  
// Convert the string "10" to the numeric value 10.  
int vCount = Convert.ToInt32(vCountString) ;  
// Convert the numeric value 10 into the string "10".  
vCountString = Convert.ToString(vCount) ;
```

The second step (turning a number into a string) will always work. The first step (turning a string into a number) won't work if the string contains letters or non-numeric characters, in which case an error will occur. Chapter 8 describes how you can use error handling to detect and neutralize this sort of problem.

The Convert class is a good all-purpose solution, but you'll also find other static methods that can do the work, if you dig around in the .NET library.

The following code uses the static Int32.Parse() method to perform the same task:

```
int vCountString = "10" ;  
// Convert the string "10" to the numeric value 10.  
vCount = Int32.Parse(vCountString) ;
```

You'll also find that you can use objects to perform some conversions a little more elegantly. The next section demonstrates this approach with the ToString() method.

4.6 OBJECT-BASED MANIPULATION

.NET is object-oriented to the core. In fact, even ordinary variables are really full-fledged objects in disguise. This means that common data types have the built-in smarts to handle basic operations (such as counting the number of characters in a string). Even better, it means you can manipulate strings, dates, and numbers in the same way in C# or VB language.

As an example, every type in the .NET class library includes a ToString() method. The default implementation of this method returns the class name. In simple variables, a more useful result is returned: the string representation of the given variable.

The following code snippet demonstrates how to use the ToString() method with an integer:

```
string vMyString;  
int vMyInteger = 100 ;  
  
// Convert a number to a string. vMyString will have the contents "100".  
vMyString = vMyInteger.ToString( ) ;
```

To understand this example, you need to remember that all int variables are based on the Int32 type in the .NET Class library. The ToString() method is built in the Int32 Class, so it's available when you use an integer in any language.

The next few sections explore the object-oriented underpinnings of the .NET data types in more detail.

4.6.1 The String Object

One of the best examples of how class members can replace built-in functions is found with strings. In the past, every language has defined its own specialized functions for strings manipulation. In .NET, however, you use the methods of the String Class, which ensures consistency between all .NET languages.

The following code snippet shows several ways to manipulate a string using object nature:

```
string vMyString = "This is a test string" ;
vMyString = vMyString.Trim( ) ;           // = "This is a test string"
vMyString = vMyString.Substring(0, 4) ;    // = "This"
vMyString = vMyString.ToUpper( ) ;         // = "THIS"
vMyString = vMyString.Replace("IS", "AT") ; // = "THAT"
int lenght = vMyString.Length;             // = 4
```

The first few statements use built-in methods, such as Trim(), SubString(), ToUpper(), and Replace(). These methods generate new strings, and each of these statements replaces the current vMyString with the new string object. The final statement uses a built-in Length property; which returns an integer that represents the number of characters in the string.

Note that the SubString() method requires a starting offset and a character length. Strings use zero-based counting. This means that the first letter is in position 0, the second letter is in position 1, and so on. You'll find this standard of zero-based counting throughout the .NET Framework for the sake of consistency. You've already seen it at work with arrays.

You can even use the string methods in succession in a single (rather ugly) line:

```
string vMyString = "This is a test string" ;
vMyString = vMyString.Trim( ).Substring(0, 4).ToUpper( ).Replace("IS", "AT") ;
Or, to make life more interesting, you can use the string methods on strings literals just
as easily as string variables:
vMyString = "hello".ToUpper();           // Sets vMyString to "HELLO"
```

Table 4.6.1 lists some useful members of the System.String Class

Member	Description
Length	Returns the numbers of characters in the string (as an integer).
ToUpper() and ToLower()	Returns a copy of the string with all the characters changed to uppercase or lowercase characters.
Trim(), TrimEnd(), and TrimStart()	Removes spaces or some other characters from either (or both) ends of a string
PadLeft() and PadRight()	Adds the specified character to the appropriate side of a string, as many times as necessary to make the total length of the string equal to number you specify. For example, "Hi".PadLeft(5, '@') returns the string "@@ @Hi"
Insert()	Puts another string inside a string at a specified (zero-based) index position. For example, Insert(1, "pre") adds the string <i>pre</i> after the first character of the current string
Remove()	Removes a specified number of string from a position. For example, Remove(0, 1) removes the first character.
Replace()	Replaces a specified substring with another string. For example, Remove(0, 1) removes the first character.
Substring()	Extracts a portion of a string of the specified length at the specified location (as a new string). For example, Substring(0, 2) retrieves the first two characters.
StartsWith() and EndsWith()	Determines whether a string starts or ends with a specified substring. For example, StartsWith("pre") will return either true or false,

	depending on whether the strings begins with the letters <i>pre</i> in lowercase
IndexOf() and LastIndexOf()	Finds the zero-based position of a substring in a string. This returns only the first match and can start at the end or beginning. You can also use overloaded versions of these methods that accept a parameter that specifies the position to start the search.
Split()	Divides a string into an array of substrings delimited by a specific substring. For example, with Split(".") you could chop a paragraph into an array of sentence strings.
Join()	Fuses an array of strings into a new string. You can also specify a separator that will be inserted between each element.

4.6.2 Date Time Object and TimeSpan Types Object

The DateTime and TimeSpan data types also have built-in methods and properties. These class members allow you to perform three useful tasks:

1. Extract a part of a DateTime (for example, just the year) or convert a TimeSpan to a specific representation (such as the total number of days or total number of minutes).
2. Easily Perform date calculations.
3. Determine the current date and time and other information (such as the day of the week or whether the date occurs in a leap year).

For example, the following block of code creates a DateTime object, sets it to the current date and time, and adds number of days. It then creates a string that indicates the year that the new date falls in (for example, 2008):

```
DateTime vMyDate = DateTime.Now ;
vMyDate = vMyDate.AddDays(100) ;
string vDateString = vMyDate.Year.ToString( ) ;
```

The next example shows how you can use a TimeSpan object to find the total number of minutes between two DateTime objects:

```
DateTime vMyDate1 = DateTime.Now ;
DateTime vMyDate2 = DateTime.Now.AddHours(3000) ;
```

```
TimeSpan vDifference ;
vDifference = vMyDate2.Subtract(vMyDate1) ;
```

```
double vNumberOfMinutes ;
vNumberOfMinutes = vDifference.TotalMinutes ;
```

The DateTime and TimeSpan Classes also support the + and - arithmetic operators, which do the same work as build-in methods. That means you can rewrite the example show earlier like this:

```
// Adding a TimeSpan to a DateTime creates a new DateTime.
```

```
DateTime vMyDate1 = DateTime.Now ;
TimeSpan vInterval = TimeSpan.FromHours(3000) ;
DateTime vMyDate2 = vMydate1 + vInterval ;
```

```
// Subtracting one DateTime object from another produces a TimeSpan.
```

```
TimeSpan vDifference;
vDifference = vMyDate2 - vMyDate1 ;
```

These examples give you an idea of the flexibility .NET provides for manipulating date and time data.

Tables 4.6.2 Lists some of the more useful built-in features of the DateTime.

Member	Description
Now	Gets the current date and time. You can also use the <code>UtcNow</code> property to take the current computer's time zone into account. <code>UtcNow</code> gets the time as a <i>Universal Time Coordinated</i> (UTC). Assuming your computer is correctly configured, this corresponds to the current time in the Western European (UTC+0) time zone.
Today	Gets the current date and leaves time set to 00.00.00.
Year, Date, Month, day, Hour, Minute, Second, and Millisecond	Returns one part of the <code>DateTime</code> object as an integer. For example, <code>Month</code> will return 12 for any day in December.
DayOfWeek	Returns an enumerated value that indicates the day of the week of the week for this <code>DateTime</code> , using the <code>DayOfWeek</code> enumeration. For example, if the date falls on Sunday, this will return <code>DayOfWeek.Sunday</code> .
Add() and Subtract()	Adds an integer that represents a number of years, months, and so on, and returns a new <code>DateTime</code> . You can use a negative integer to perform a date subtraction.
AddYears(), AddMonths(), AddDays(), AddHours(), Addminutes(), AddSeconds(), AddMilliseconds()	Adds an integer that reprints a number of years, months, and so on, and returns a new <code>DateTime</code> . You can use a negative integer to perform a date subtraction.
DaysInMonth()	Returns the number of days in the specified month in the specified year.
IsLeapYear()	Returns true or false depending on whether the specified year is a leap year.
ToString()	Returns a string representation of the current <code>DateTime</code> object. You can also use an overloaded version of this method that allows you to specify a parameter with a format string.

Tables 4.6.3 Lists some of the more useful built-in features of the TimeSpan object.

Member	Description
Days, Hours, Minutes, Seconds, Milliseconds	Returns one component of the current <code>TimeSpan</code> . For example, the <code>Hours</code> property can return an integer from -23 to 23.
TotalDays, TotalHours, TotalMinutes, TotalSeconds, TotalMilliseconds	Returns the total value of the current <code>TimeSpan</code> as a number of days, hours, minutes, and so on. The value is returned as a double, which may include a fractional value. For example, the <code>TotalDays</code> property might return a number like 234.324.
Add() and Subtract() ;	Combines <code>TimeSpan</code> objects together.
FromDays(), FromHours(), FromMinutes(), FromSeconds(), FromMilliseconds()	Allows you to quickly create a new <code>TimeSpan</code> . For example, you can use <code>TimeSpan.FromHours(24)</code> to create a <code>TimeSpan</code> object exactly 24 hours long.
ToString()	Returns a string representation of the current <code>TimeSpan</code> object. You can also use an overloaded version of this method that allows you to specify a parameter with a format string

4.6.3 The Array Object

Arrays also behave like objects in the new world of .NET. For example, if you want to find out the size of a one-dimensional array, you can use the `Length` property or the `GetLength()` method, both of which return the total number of elements in an array.

```
int [ ] vMyArray = {1, 2, 3, 4, 5};
int vNumberOfElements;

vNumberOfElements = vMyArray.Length;           // vNumberOfElements = 5.
```

You can also use the `GetUpperBound()` method to find the highest index number in an array. The following code snippet shows this technique in action:

```
int [ ] vMyArray = {1, 2, 3, 4, 5};
int vBound;

// Zero represents the first dimension of an array.
vBound = vMyArray.GetUpperBound(0);           // vBound = 4
```

On a one-dimensional array, `GetUpperBound()` always returns a number that's one less than the length. That's because the first index number is 0, and the last index number is always one less than the total number of items. However, in a two-dimensional array, you can find the highest index number for a specific dimension in an array. For example, the following code snippet uses `GetUpperBound()` to find the total numbers of rows and the total number of columns in a two-dimensional array:

```
// Create a 4 x 2 array (a grid with four rows and two columns).
int [ , ] vIntArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};

int vRows = vIntArray.GetUpperBound(0);       // Rows = 4
int vColumns = vIntArray.GetUpperBound(1);    // Columns = 2
```

Having these values, the array length and indexes, is handy when looping through the contents of an array, as you'll see later in this chapter, in the "Loops" section. Arrays also provide a few other useful methods, which allow you to sort them, reverse them, and search them for a specified element. Table 2-6 lists some useful members of the `System.Array` Class.

Member	Description
<code>Length</code>	Returns an integer that represents the total number of elements in all dimensions of an array. For example, an array 3 x 3 array has a length of 9.
<code>GetLowerBound()</code> and <code>GetUpperBound()</code>	Determines the dimensions of an array. As with just about everything in .NET, you start counting at zero (which represents the first dimension).
<code>Clear()</code>	Empties part of all of an array's contents, depending on the index values that you supply. The elements revert to their initial empty values (such as 0 for numbers).
<code>IndexOf()</code> and <code>LastIndexOf()</code>	Searches a one-dimensional array for a specified value and returns the index number. You cannot use this with Multidimensional arrays.
<code>Sort()</code>	Sorts a one -dimensional array made up of comparable data such as strings or numbers.
<code>Reverse()</code>	Reverses a one-dimensional array so that its elements are backwards, from last to first.

Check your progress 4.6

Answer the following questions.

a) What do you mean by variable operation?

.....

.....

b) What is a Type conversion?

.....

.....

c) What is Object-based Manipulation?

.....

.....

4.7 CONDITIONAL STRUCTURES

In many ways, conditional logic (deciding which action to take based on user input, external conditions, or other information), is the heart of programming. All conditional logic starts with a *condition*: a simple expression that can be evaluated to true or false. Your code can then make a decision to execute different logic depending on the outcome of the condition. To build a condition, you can use any combination of literal values or variables along with *logical operators*. Table lists the basic logical operators.

Operator	Description
= =	Equal to...
! =	Not Equal to...
<	Less than...
>	Greater than...
< =	Less than or equal to...
> =	Greater than or equal to...
& &	Logical and (evaluates to true only if both expressions are true). If the first expression is false, the second expression is not evaluated.
	Logical or (evaluates to true only if both expressions are true). If the first expression is true, the second expression is not evaluated.

You can use all the comparison operators with any numeric types. With string data types, you can use only the equality operators (== and !=). C# doesn't support other types of string comparison operators. Instead, you need to use the `String.Compare()` method. The `String.Compare()` method deems that a string is "less than" another string if it occurs earlier in an alphabetic sort. Thus, *apple* is less than *attach*. The return value from `String.Compare` is 0 if the strings match, 1 if the first supplied string is greater than the second, and -1 if the first string is less than the second. Here's an example:

```
int vResult ;
vResult = String.Compare("apple", "attach") ;    // Result = -1
vResult = String.Compare("apple", "all") ;       // Result = 1
vResult = String.Compare("apple", "apple") ;     // Result = 0

// Another way to perform string comparisons.
string word = "apple" ;
vResult = Word.CompareTo("attach") ;             // Result = -1
```

The if Statement

If statement is the powerhouse of conditional logic, able to evaluate any combination of conditions and deal with multiple and different pieces of data. Here's an example with if statement that features two conditions:


```

if (vMyNumber > 10) ;
{
    // Do Something.
}
else if (vMyString == "hello")
{
    // Do Something.
}
else
{
    // Do Something.
}

```

Keep in mind that If construct matches one condition at most. For example, If vMyNumber is greater than 10, the first condition will be met. That means the code in the first conditional block will run, and no other conditions will be evaluated. If block can have any numbers of conditions. If you test only a single condition, you don't need to includes any else blocks.

The switch statement

C# also provides a switch statement that you can use to evaluated a single variable or expression for multiple possible values. The only limitation is that the variable you're evaluating must be an integer-based data type, a bool, a char, a string, or a value from an enumeration. Other data types aren't supported.

In the following code, each case examines the vMyNumber variable and tests whether it's equal to a specific integer:

```

switch (vMyNumber)
{
    case 1 :
        // Do Something
        break ;
    case 2 :
        // Do Something
        break ;
    default :
        // Do Something
        break ;
}

```

You'll notice that in C# syntax inherits the convention of C/C++ programming, which requires that every branch in a switch statement be ended by a special break keyword. If you omit this keyword, the compiler will alert you and refuse to build your application. The only exception is if you choose to stack multiple case statements directly on top of each other with no intervening code. This allows you to write one segment of code that handles more than one case. Here's an example:

```

switch (vMyNumber)
{
    case 1 :
    case 2 :
        // This codes executes if vMyNumber is 1 or 2.
        break ;
    default :
        // Do Something
        break ;
}

```

Unlike, If statement, the switch statement is limited to evaluating a single piece of information at a time. However, it provides a leaner, clearer syntax than If statement for situations in which you need to test a single variable.

4.8 LOOP STRUCTURES

Loops allow you to repeat a segment of code multiple times. C# has three basic types of loops. You choose the type of loop based on the type of task you need to perform. Your choices are as follows:

- You can loop a set of number of times a For loop.
- You can loop through all the items in a collection of data using a foreach loop.

- You can loop while a certain condition holds true, using a while loop.

For and foreach loops are ideal for chewing through sets of data that have known, fixed sizes. The while loop is a more flexible construct that allows you to continue processing until a complex condition is met. The while loop is often used with repetitive tasks or calculating things that don't have a set number of iterations.

The for loop

For loop is a basic ingredient in many programs. It allows you to repeat a block of code a set number of times, using a built-in counter. To create For loop, you need to specify a starting value, an ending value, and the amount to increment with each pass. Here's one example:

```
for (int v = 0 ; v < 10 ; v ++ )
{
    // This code executes ten times.
    System.Diagnostics.Debug.Write(v) ;
}
```

You'll notice that the for loop starts with brackets that indicate three important pieces of information. The first portion, (int v = 0), creates the counter variable (v) and sets its initial value (0). The third portion, (v ++), increments the counter variable (v). In this example, the counter is incremented by 1 after each pass. That means v will be equal to 0 for the first pass, equal to 1 for the second pass, equal to 2 for the third pass, and so on. However, you could adjust this statement so that it decrements the counter (or performs any other operation you want). The middle portion, (v < 10), specifies the condition that must be met for the loop to continue. This condition is tested at the start of every pass through the block. If v is greater than or equal to 10, the condition will evaluate to false, and the loop will end.

If you run this code using a tool such as Visual Studio, it will write the following numbers in the Debug window:

```
0 1 2 3 4 5 6 7 8 9
```

It often makes sense to set the counter variable based on the number of times you're processing. For example, you can use for loop to step through the elements in an array by checking the size of the array before you begin. Here's the code you would use:

```
string[] vStringArray = {"one", "two", "three"};

for (int v = 0 ; v < vStringArray.Length ; v ++ )
{
    System.Diagnostics.Debug.Write(vStringArray [ v ] + " " ) ;
}
```

Sample Output

```
one two three
```

The foreach Loop

BLOCK-LEVEL SCOPE

If you define a variable inside some sort of block structure (such as a loop or conditional block), the variable is automatically released when your code exits the block. That means you will no longer be able to access it.

The following code demonstrates this behavior:

```
int vTempVariableA ;
for (int v = 0 ; v < 10 ; v ++ )
{
    int vTempVariableB ;

    vTempVariableA = 1 ;
    vTempVariableB = 1 ;
}

// You cannot access vTempVariableB here.
// However, you can still access vTempVariableA here.
```

This change won't affect many programs. It's really designed to catch a few more accidental errors. If you do need to access a variable inside and outside of some type of block structure, just define the variable *before* the block starts.

C# also provides a foreach loop that allows you to loop through the items in a set of data. With a foreach loop, you don't need to create an explicit counter variable. Instead, you create a variable that represents the type of data for which you're looking. Your code will then loop until you've had a change to process each piece of data set. The foreach loop is particularly useful for traversing the data in collections and arrays. For example, the next code segment loops through the items in an array using foreach. This code has exactly the same effect as the previous example but is a little simpler:

```
string[] vStringArray = {"one", "two", "three"} ;

foreach (string vElement in vStringArray)
{
    // This code loops three times, with the element variable set to
    // "one", then "two", and then "three".
    System.Diagnostics.Debug.Write( vElement + " " ) ;
}
```

In this case, the foreach loop examines each item in the array and tries to convert it to a string. Thus, the foreach loop defines a string variable named element. If you used a different data type, you'd receive an error.

The foreach loop has one key limitation: it's read only. For example, if you wanted to loop through an array and change the values in that array at the same time, foreach code wouldn't work. Here's an example of some flawed code:

```
int[] vIntArray = {1, 2, 3} ;

foreach (int vNumber in vIntArray)
{
    vNumber += 1 ;
    // This code executes ten times.
}
```

In this case, you would need to fall back on a basic for loop with a counter.

The while loop

Finally, C# supports a while loop that tests a specific condition before or after each pass through the loop. When this condition evaluates to false, the loop is exited.

Here's an example that loops ten times. At the beginning of each pass, the code evaluates whether the counter(v) is less than some upper limit (in this case, 10). If it is, the loop performs iteration.

```
int v = 0;

while (v < 10)
```

```

{
    v += 1 ;
    // This code executes ten times.
}
int v = 0 ;

do
{
    v += 1 ;
    // This code executes ten times.
}
while ( v < 10) ;

```

Both of these examples are equivalent, unless the condition you're testing is false to start. In that case, the while loop will skip the code entirely. They do...while loop, on the other hand, will always execute the code at least once, because it doesn't test the condition until the end.

4.9 FUNCTIONS & SUBROUTINES

Methods are the most basic building block you can use to organize your code. Essentially, a method is a named grouping of one or more lines of code. Ideally, each method will perform a distinct, logical task. By breaking your code down into methods, you not only simplify your life, but you also make it easier to organize your code into classes and step into the world of object-oriented programming.

The first decision you need to make when declaring a method is whether you want to return any information. For example, a method named `GetStartTime()` might return a `DateTime` object that represents the time an application was first started. A method can return, at most, one piece of data.

When you declare a method in C#, the first part of the declaration specifies the data type of the return value, and the second part indicates the method name. If your method doesn't return any information, you should use the `void` keyword instead of a data type at the beginning of the declaration.

Here are two examples, one method that doesn't return anything, and one that does:

// This method doesn't return any information.

```

void MyMethodNoReturnData()
{
    // Code goes here.
}
// This method returns an integer.

```

```

int MyMethodReturnsData( )
{
    // As example, return the number 10.
    return 10 ;
}

```

Notice that the method name is always followed by parentheses. This allows the compiler to recognize that it's a method.

In this example, the methods don't specify their accessibility. This is just common C# convention. You're free to add an accessibility keyword (such as `public` or `private`) as show here:

```

private void MyMethodNoReturnData()
{
    // Code goes here.
}

```

The accessibility determines how different classes in your code can interact. Private methods are hidden from view and are available only locally, whereas public methods can be called by all the other classes in your application. To really

understand what this means, you'll need to read the next chapter, which discusses accessibility in more detail.

Invoking your methods is straightforward; you simply type the name of the method, followed by parentheses. If your method returns data, you have the option of using the data it returns or just ignoring it:

```
// This call is allowed.
MyMethodNoReturnedData( ) ;
// This call is allowed.
MyMethodReturnsData( ) ;
// This call is allowed.
int vMyNumber ;
vMyNumber = MyMethodReturnsData( ) ;
// This call isn't allowed.
// MyMethodNoReturnedData( ) does not return any information.
vMyNumber = MyMethodNoReturnedData( ) ;
```

4.9.1 Parameters

Methods can also accept information through parameters. Parameters are declared in a similar way to variables. By convention, parameter names always begin with a lowercase letter in any language.

Here's how you might create a function that accepts two parameters and returns their sum:

```
private int AddNumber(int pNumber1, int pNumber2)
{
    return pNumber1 + pNumber2 ;
}
```

When calling a method, you specify any required parameters in parentheses or use an empty set of parentheses if no parameters are required:

```
// Call a method with no parameters.
MyMethodNoReturnedData( ) ;
// Call a method that requires two integer parameters.
MyMethodNoReturnedData(10, 20) ;
// Call a method with two integer parameters and an integer value.
int vReturnValue = AddNumber(10, 20) ;
```

4.9.2 Method Overloading

C# supports method *overloading*, which allows you to create more than one method with the same name, but with a different set of parameters. When you call the method, the CLR automatically chooses the correct version by examine the parameters you supply.

This technique allows you to collect different versions of several methods together. For example, you might allow a database search that returns an array of Product objects representing records in the database. Rather than create three methods with different names depending on the criteria, such as GetAllProducts(), GetProductsInCategory(), and GetActiveProducts(), you could create three versions of the GetProducts() method. Each method would have the same name but a different *signature*, meaning it would require different parameters. This example provides two overloaded version for The GetProductPrice().

```
private decimal GetProductPrice(int ID)
{
    // Code here. }
private decimal GetProductPrice(string name)
{
    // Code here. }
// And so on...
```

Now you can look up product prices based on the unique product ID or the full product name, depending on whether you supply an integer or string argument:

```
// Get price by product ID (the first version).
vPrice = GetProductPrice(1001) ;
```

```
// Get price by product ID (the second version).  
vPrice = GetProductPrice("DVD Player") ;
```

You cannot overload a method with versions that have the same signature, that is, the same number of parameters and parameters data types, because the CLR will not be able to distinguish them from each other. When you call an overloaded method, the version that matches the parameter list you supply is used. If no version matches, an error occurs.

4.9.3 Delegates

Delegates allow you to create a variable that "points" to a method. You can use this variable at any time to invoke the method. Delegates help you write flexible code that can be reused in many situations.

The first step when using a delegate is to define its *signature*. The signature is a combination of several pieces of information about a method: its return type, the number of parameters it has, and the data type of each parameter.

A delegate variable can point only to a method that matches its specific signature. In other words, the methods must have the same return type, the same number of parameters, and the same data type for each parameter as the delegate. For example, if you have a method that accepts a single string parameter and another method that accepts two string parameters, you'll need to use a separate delegate type for each method.

To consider how this works in practice, assume your program has the following method:

```
Private string TransLateEnglishToFrench(string English)  
{  
    // Code goes here.  
}
```

This method accepts a single string argument and returns a string. With those two details in mind, you can define a delegate that matches this signature. Here's how you would do it:

```
Private delegate string StringFunction(string InputString) ;
```

Notice that the name you choose for the parameters and the name of the delegate don't matter. The only requirement is that the data types for the return value and parameters match exactly.

Once you've defined a type of delegate, you can create and assign a delegate variable like this:

```
StringFunction vFunctionReference ;
```

Once you have a delegate variable, the funs begin. Using your delegate variable, you can point to any method that has the matching signature. In this example, the StringFunction delegate type requires one string parameter and returns a string. Thus, you can use the functionreference variable to store a reference to the TransLateEnglishToFrench() method you saw earlier. Here's how to do it:

```
vFunctionReference = TransLateEnglishToFrench;
```

Now that you have a delegate variable that references a method, you can invoke the method *through* the delegate. To do this, you just use the delegate name as though it were the method name:

```
string frenchString ;  
frenchString = functionreference("Hello") ;
```

In the previous code example, the method that the functionReference delegate points to will be invoked with the parameter value "Hello", and return value will be stored in the frenchString variable.

The following code shows all these steps, creating a delegate variable, assigning a method, and calling the method, from start to finish:

```
// Create a delegate variable.
StringFunction functionReference ;
// Store a reference to a matching method in the delegate.
functionReference = TransLateToEnglishToFrench ;
// Run the method that functionReference points to.
// In this case, it will be TransLateEnglishToFrench( ) .
string frenchString = functionReference("Hello") ;
```

The value of delegates is in the extra layer of flexibility they add. It's not apparent in this example, because the same piece of code creates the delegate variable, and another method would use it. The benefit in this scenario is that the second method doesn't need to know where the delegate points. Instead, it's flexible enough to use any method that has the right signature. In the current example, imagine a translation library that could translate between English and a variety of different languages, depending on whether the delegate it uses points to TranslateEnglishToFrench(), TranslateEnglishToSpanish(), TranslateEnglishToGerman(), and so on.

Check your progress 4.9

Answer the following questions in 1-2 sentences.

a) What is delegate?

.....

b) What is conditional structure?

.....

c) What is method overloading?

.....

4.10 SUMMARY

- Variables can store numbers, text, dates, and times, and they can even point to full-fledged objects.
- Arrays allow you to store a series of values that have the same data type.
- An enumeration is a group of related constants; each of which is given a descriptive name. Each value in an enumeration corresponds to a preset integer.
- Converting information from one data type to another is a fairly common programming task.
- Loops allow you to repeat a segment of code multiple times.

4.11 CHECK YOUR PROGRESS - ANSWERS

4.2

1. Byte, int, float double etc
2. String is a collection of character, you can say that character array. Escape characters are defined with a special meaning and it start with '\ ' slash.

4.3 – 4.4

1. Collection of data having similar data types.
2. It is a group or collection of related constant.

4.5 – 4.6

1. Operation on any variable like addition, subtraction etc.
2. If we temporary change the data type of any variable then it is called type conversion.

3. C# and ASP are the object base language that means if we create a variable of int then it means that variable is a member of class integer.

4.7 – 4.9

1. Delegate are the type safe function pointer, means they can invoke a function indirectly by using referencing to the function, it is helpful reuse of our single function in many classes / program.
2. Conditional structure comprises of if else and switch etc, they get run on the basis of condition, whether is it true or false.
3. Method overloading designed to give a common name to method but, it have restriction i.e. signature of function will be different using overloading we less the work of programmer.

4.12 QUESTIONS FOR SELF - STUDY

1. What is Array? Explain with examples.
2. Explain Enumeration with program.
3. Explain Variable operation.
4. Explain Loop structure.
5. What is function? Explain it.

4.13 SUGGESTED READINGS

1. Programming C#: Building .NET Applications with C# By Jesse Liberty
2. Learning C# 2005 by Jesse Liberty
3. Microsoft MSDN
4. Beginning ASP.NET 4.5 in C#, By Matthew MacDonald
5. http://www.w3schools.com/aspnet/aspnet_arraylist.asp



[illegible]

NOTES

[illegible]

Chapter 5

ASP.NET Applications

5.0 Objectives
5.1 Introduction
5.2 ASP.NET File Types
5.3 The Application/ Bin Directory
5.4 Code-Behinds in ASP .NET
5.5 The Global.asax
5.6 ASP .NET Configuration
5.7 Summary
5.8 Check your Progress - Answers
5.9 Questions for Self – Study
5.10 Suggested readings

5.0 OBJECTIVES

After studying this chapter you will be able to –

- describe asp.net file types
- explain the use of bin directory
- discuss code-behinds
- describe global.asax
- explain the Asp.Net configuration.

5.1 INTRODUCTION

Web site applications can contain a number of file types, some supported and managed by ASP.NET, and others supported and managed by the IIS server. Most of the ASP.NET file types can be automatically generated using the Add New Item menu item in Visual Web Developer.

5.2 ASP.NET FILE TYPES

File types are mapped to applications using application mappings. For example, if you use double-click a .txt file in Windows Explorer, Notepad will probably open, because in Windows, .txt file types are mapped by default to Notepad.exe. In Web applications, file types are mapped to application extensions in IIS.

File Types Managed by ASP.NET

File type	Location	Description
.asax	Application root.	Typically a Global.asax file that contains code that derives from the HttpApplication class. This file represents the application and contains optional methods that run at the start or end of the application lifetime.
.ascx	Application root or a subdirectory.	A Web user control file that

		defines a custom, reusable control.
.ashx	Application root or a subdirectory.	A generic handler file that contains code that implements the IHttpHandler interface.
.asmx	Application root or a subdirectory.	An XML Web services file that contains classes and methods that are available to other Web applications by way of SOAP.
.aspx	Application root or a subdirectory.	An ASP.NET Web forms file (page) that can contain Web controls and presentation and business logic.
.axd	Application root.	A handler file used to manage Web site administration requests, typically Trace.axd.
.browser	App_Browsers subdirectory.	A browser definition file used to identify the features of client browsers. .
.cd	Application root or a subdirectory.	A class diagram file.
.compile	Bin subdirectory.	A precompiled stub file that point to an assembly representing a compiled Web site file. Executable file types (.aspx, ascx, .master, theme files) are precompiled and put in the Bin subdirectory.
.config	Application root or a subdirectory.	A configuration file (typically Web.config) containing XML elements that represent settings for ASP.NET features. .
.cs, .jsl, .vb	App_Code subdirectory, or in the case of a code-behind file for an ASP.NET page, in the same directory as the Web page.	Class source-code file that is compiled at run time. The class can be an HTTP Module , an HTTP Handler , a code-behind file for an ASP.NET page, or a stand-alone class file containing application logic.
.csproj, .vbproj, vjsproj	Visual Studio project directory.	A project files for a Visual Studio client-application project. .
.disco, .vsdisco	App_WebReferences subdirectory.	An XML Web services discovery file used to help locate available Web services.
.dll	Bin subdirectory.	A compiled class library files (assembly). Note that instead of placing compiled

		assemblies in the Bin subdirectory, you can put source code for classes in the App_Code subdirectory.
.licx, .webinfo	Application root or a subdirectory.	A license file. Licensing allows control authors to help protect intellectual property by checking that a user is authorized to use the control.
.master	Application root or subdirectory.	A master page that defines the layout for other Web pages in the application.
.mdb, .ldb	App_Data subdirectory.	An Access database file.
.mdf	App_Data subdirectory.	SQL database file for use with SQL Server Express.
.msgx, .svc	Application root or a subdirectory.	An Indigo Messaging Framework (MFx) service file.
.rem	Application root or a subdirectory.	A removing handler file.
.resources, .resx	App_GlobalResources or App_LocalResources subdirectory.	A resource file that contains resource strings that refer to images, localizable text, or other data.
.sdm, .sdmDocument	Application root or a subdirectory.	A system definition model (SDM) file.
.sitemap	Application root.	A site-map file that contains the structure of the Web site. ASP.NET comes with a default site-map provider that uses site-map files to easily display a navigational control in a Web page.
.skin	App_Themes subdirectory.	A skin file containing property settings to apply to Web controls for consistent formatting.
.sln	Visual Web Developer project directory.	A solution file for a Visual Web Developer project.
.soap	Application root or a subdirectory.	A SOAP extension file.

Source : [http://msdn.microsoft.com/en-IN/library/2wawkw1c\(v=vs.80\).aspx](http://msdn.microsoft.com/en-IN/library/2wawkw1c(v=vs.80).aspx)

File Types Managed by IIS

File type	Location	Description
.asa	Application root.	Typically a Global.asax file that contains optional methods that runs at the start or end of the ASP session or application lifetime. .
.asp	Application root or a subdirectory.	An ASP Web page that contains @ directives and script code that uses the ASP built-in objects.
.cdx	App_Data subdirectory.	A compound index file structure file for Visual FoxPro.
.cer	Application root or a subdirectory.	A certificate file used to authenticate a Web site.

.idc	Application root or a subdirectory.	An Internet Database Connector file mapped to httpodbc.dll.
.shtm, .shtml, .stm	Application root or a subdirectory.	Mapped to ssinc.dll.

Static File Types :

IIS serves static files only if their file-name extensions are registered in the MIME types list. This list is stored in the MimeMap IIS metabase property for an application. If a file type is mapped to an application extension, it does not need to be included in the MIME types list unless you want the file to be treated like a static file. Typically, ASP.NET source code file types should not be in the MIME types list because that might allow browsers to view the source code.

The following table lists only a few of the registered file types.

File type	Location	Description
.css	Application root or subdirectory, or App_Themes subdirectory.	Style sheet files used to determine the formatting of HTML elements.
.htm, .html	Application root or subdirectory.	Static Web files written in HTML code.

Check your progress 5.2

Answer the following questions in 1-2 sentences.

a) Write any two asp.net file types.

.....
.....

b) What is static file type?

.....
.....

5.3 THE APPLICATION /BIN DIRECTORY

A problem with using the COM model for Web application components is that those components must be registered before they can be used from a traditional ASP application. Remote administration of these types of applications is often not possible, because the registration tool must be run locally on the server. To make matters more difficult, these components remain locked on disk once they are loaded by an application, and the entire Web server must be stopped before these components can be replaced or removed.

ASP.NET attempts to solve these problems by allowing components to be placed in a well-known directory, to be automatically found at run time. This well-known directory is always named **/bin**, and is located immediately under the root directory for the application (a virtual directory defined by Internet Information Services (IIS)). The benefit is that no registration is required to make components available to the ASP.NET Framework application components can be deployed by simply copying to the /bin directory or performing an FTP file transfer.

In addition to providing a zero-registration way to deploy compiled components, ASP.NET does not require these components to remain locked on disk at run time. Behind the scenes, ASP.NET duplicates the assemblies found in /bin and loads these "shadow" copies instead. The original components can be replaced even while the Web server is still running, and changes to the /bin directory are

automatically picked up by the runtime. When a change is detected, ASP.NET allows currently executing requests to complete, and directs all new incoming requests to the application that uses the new component or components.

5.4 CODE BEHINDS IN ASP.NET

Even though ASP.NET is the next version of ASP, it is more than just a next version. ASP.NET is completely re-designed from the ground up and it provides a neat object oriented programming model. An ASP.NET page is an object derived from the .NET Page class. We all know that ASP.NET provides several new features and one of the important features is the separation of code and content.

In today's ASP application, we have a mix of HTML and Scripts making the code difficult to read, debug and maintain. ASP.NET relieves us from this problem by promoting the separation of code and content. That is, the user interface and the user interface programming logic need not necessarily be written in a single page. There are two ways in which you can separate the code and content:

- 1) By using Code-behind files that are pre-compiled modules written in any of the Microsoft .NET runtime compliant languages.
- 2.) By developing the frequently used page logic in to separate files called Pagelets (also known as page controls) and by using them several times in your ASP.NET pages.

As the title of the article indicates, I will explain the first mechanism that is used to neatly encapsulate the UI functionality in to a separate pre-compiled module. You will typically follow the steps below to develop an ASP.NET page that uses code-behind pages:

- 1) Create the User Interface using the HTML and/or Web controls and store them in an ASP.NET page with extension ".aspx".
- 2) Create the code-behind file using the .NET runtime classes that mimic the entire ASP.NET page with the member variables that correspond to the HTML and/or Web controls used in the ASP.NET page. The extension of code behind files will vary depending on the language you use. Typical extensions will be .cs for C#, .vb for VB7 and .js for JScript .NET

Now, we know that there are two separate files: one for the UI and the other for the UI logic. But, how do we relate these two files so that the ASP.NET page compiler will locate the code-behind file for the ASP.NET page. The glue between these two files is provided through the Page Directives, which are used to specify optional settings at the page level.

Creating the "Content"

Enough theory said about code-behind files. Let us get our hands on to the actual coding of a code-behind file. The ASP.NET page that displays the Web UI can be seen below. It uses a post-back form to query the user for username and information:

```
<%@ Page language="c#" Codebehind="CodeBehind.cs"
Inherits="ASPPlus.CodeBehind" %>
```

```
<HTML><BODY>
<form action="CodeBehind.aspx" method=post runat="server">
```

```
  <p align=center>Demonstration of Code Behind File</P>
  <p align=center><asp:Label id=Message runat="server"></asp:Label></P>
  <p>
```

```
<h5>Sign-In Form</h5>
Login: <asp:TextBox id=txtLogin Width=200 Runat=server />
<asp:RequiredFieldValidator ControlToValidate="txtLogin"
  Display="Static" Font-Name="verdana,arial,sans-serif"
  Font-Size="9pt" ErrorMessage="You must enter a Login."
  runat="server" ID="RFVLogin" />
```

```

<P>
Password: <asp:TextBox id=txtPassword Width=200 Runat=server
        TextMode="Password" />
<asp:RequiredFieldValidator ControlToValidate="txtPassword"
        Display="Static" Font-Name="verdana,arial,sans-serif"
        Font-Size="9pt" ErrorMessage="You must enter a password."
        runat=server ID="RFVPassword"> </asp:RequiredFieldValidator>

<P>
<asp:Button id=btnSignIn Runat="server"
        Text="Sign In" onClick="btnSignIn_Click" />
</FORM>
</body></html>

```

The first line of the ASP.NET page is the page directive that specifies the name of the code behind file and the actual class name inside that file. In our example, the file name is **CodeBehind.cs** and the class name is **ASPPlus.CodeBehind**.

```

<%@ Page language="c#" Codebehind="CodeBehind.cs"
Inherits="ASPPlus.CodeBehind" %>

```

The namespace that I used for this code behind page is **ASPPlus**. Namespaces provide the naming scope for the classes and is very convenient way to represent hierarchy of classes. The rest of the code consists of HTML tags and Web Control declaration tags. The main thing to note down in this page is the ID properties of the controls. I will explain the importance of these IDs when we discuss the code-behind file.

Check your progress 5.3 – 5.4

Answer the following questions in 1-2 sentences.

a) What is bin directory?

.....

b) Explain code behind in Asp.net.

.....

5.5 THE GLOBAL.ASAX

Introduction

The Global.asax file (also known as the ASP.NET application file) is an optional file that is located in the application's root directory and is the ASP.NET counterpart of the Global.asax of ASP. This file exposes the application and session level events in ASP.NET and provides a gateway to all the application and the session level events in ASP.NET. This file can be used to implement the important application and session level events such as Application_Start, Application_End, Session_Start, Session_End, etc. This article provides an overview of the Global.asax file, the events stored in this file and how we can perform application wide tasks with the help of this file.

What is the Global.asax file?

According to MSDN, "The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET." The Global.asax file is parsed and dynamically compiled by ASP.NET into a .NET Framework class the first time any resource or URL within its application namespace is activated or requested. Whenever the application is requested for the first time, the Global.asax file is parsed and compiled to a class that extends the HttpApplication class. When the Global.asax file changes, the framework reboots the application and the Application_OnStart event is fired once again when the

next request comes in. Note that the Global.asax file does not need recompilation if no changes have been made to it. There can be only one Global.asax file per application and it should be located in the application's root directory only.

Events in the Global.asax file

The following are some of the important events in the Global.asax file.

Application_Init : The Application_Init event is fired when an application initializes the first time.

Application_Start : The Application_Start event is fired the first time when an application starts.

Session_Start : The Session_Start event is fired the first time when a user's session is started. This typically contains for session initialization logic code.

Application_BeginRequest : The Application_BeginRequest event is fired each time a new request comes in.

Application_EndRequest : The Application_EndRequest event is fired when the application terminates.

Application_AuthenticateRequest :

The Application_AuthenticateRequest event indicates that a request is ready to be authenticated. If you are using Forms Authentication, this event can be used to check for the user's roles and rights.

Application_Error : The Application_Error event is fired when an unhandled error occurs within the application.

Session_End : The Session_End Event is fired whenever a single user Session ends or times out.

Application_End : The Application_End event is last event of its kind that is fired when the application ends or times out. It typically contains application cleanup logic.

Using the Global.asax file

The following code sample shows how we can use the events in the Global.asax file to store values in the Application state and then retrieve them when necessary. The program stores an Application and a Session counter in the Application state to determine the number of times the application has been accessed and the number of users currently accessing the application.

Example 1

```
using System;

using System.ComponentModel;
using System.Web;
using System.Web.SessionState;

public class Global : HttpApplication
{
    protected void Application_Start(Object sender, EventArgs e)
    {
        Application["appCtr"] = 0;
        Application["noOfUsers"] = 0;
    }

    protected void Application_BeginRequest(Object sender, EventArgs e)
    {
        Application.Lock();
        Application["appCtr"] = (int) Application["appCtr"] + 1;
        Application.Unlock();
    }

    protected void Session_Start(Object sender, EventArgs e)
    {

```

```

Application.Lock();
Application["noOfUsers"] = (int) Application["noOfUsers"] + 1;
Application.Unlock();
}
// Code for other handlers
}

```

After storing the values in the Application state, they can be retrieved using the statements given in the code sample below.

Example 2

```

Response.Write("This application has been
accessed "+Application["appCtr"] + " times");
Response.Write("There are "+      Application["noOfUsers"] + " users accessing this
application");

```

5.6 ASP.NET CONFIGURATION

The ASP.NET configuration system features an extensible infrastructure that enables you to define configuration settings at the time your ASP.NET applications are first deployed so that you can add or revise configuration settings at any time with minimal impact on operational Web applications and servers.

The ASP.NET configuration system provides the following benefits:

- Configuration information is stored in XML-based text files. You can use any standard text editor or XML parser to create and edit ASP.NET configuration files.
- Multiple configuration files, all named Web.config, can appear in multiple directories on an ASP.NET Web application server. Each Web.config file applies configuration settings to its own directory and all child directories below it. Configuration files in child directories can supply configuration information in addition to that inherited from parent directories, and the child directory configuration settings can override or modify settings defined in parent directories. The root configuration file named *systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Machine.config* provides ASP.NET configuration settings for the entire Web server.
- At run time, ASP.NET uses the configuration information provided by the Web.config files in a hierarchical virtual directory structure to compute a collection of configuration settings for each unique URL resource. The resulting configuration settings are then cached for all subsequent requests to a resource. Note that inheritance is defined by the incoming request path (the URL), not the file system paths to the resources on disk (the physical paths).
- ASP.NET detects changes to configuration files and automatically applies new configuration settings to Web resources affected by the changes. The server does not have to be rebooted for the changes to take effect. Hierarchical configuration settings are automatically recalculated and reached whenever a configuration file in the hierarchy is changed.
- The ASP.NET configuration system is extensible. You can define new configuration parameters and write configuration section handlers to process them.
- ASP.NET help protect configuration files from outside access by configuring Internet Information Services (IIS) to prevent direct browser access to configuration files. HTTP access error 403 (forbidden) is returned to any browser attempting to request a configuration file directly.

Check your progress 5.5-5.6

Answer the following questions in 1-2 sentences.

a) What is Global.asax?

.....
.....

b) Explain asp.net configuration.

.....
.....

5.7 SUMMARY

- Web site applications can contain a number of file types, some supported and managed by ASP.NET, and others supported and managed by the IIS server. Most of the ASP.NET file types can be automatically generated using the Add New Item menu item in Visual Web Developer.
- In asp.net directory is always named **/bin**, and is located immediately under the root directory for the application (a virtual directory defined by Internet Information Services (IIS)).
- The Global.asax file (also known as the ASP.NET application file) is an optional file that is located in the application's root directory and is the ASP.NET counterpart of the Global.asax of ASP.

5.8 CHECK YOUR PROGRESS – ANSWERS

5.2

1. .asax, .ascx
2. If the file name, extensions are register in the MIME File types.

5.3-5.4

1. To solve the traditional problem of com object, Microsoft create a well known \ bin directory to store application file at run time.
2. Meaning of it means that your page designee is distinct from your coding.

5.5 – 5.6

1. It is a file used to declare and handle application / session level events, object for ASP .NET website running on IIS web server.
2. ASP .NET configuration means now your website will work in term of – Role, Authentication, and Authorization. These all are customizable and we find them in web.config file>system.web.

ASP .NET configuration gives us easy way to manage our web page and other type of administration. We can do it by code also.

5.9 QUESTIONS FOR SELF - STUDY

1. Explain the Asp.Net file types with examples.
2. Explain global .asax events.
3. Explain Asp.Net configuration benefits.

5.10 SUGGESTED READINGS

Beginning ASP.NET 4.5 in C#, By Matthew MacDonald



NOTES

[illegible]

Overview of ADO.NET

6.0	Objectives
6.1	Introduction
6.2	ADO.NET architecture
6.3	Accessing Data using data adapters and datasets
6.4	Binding data to data bind controls
6.5	Displaying data in data grid
6.6	Summary
6.7	Check your Progress - Answers
6.8	Questions for Self – Study
6.9	Suggested Readings

6.0 OBJECTIVES

After studying in this chapter you will be able to –

- ♦ explain the ADO.NET architecture
- ♦ discuss the use of data adapters and datasets
- ♦ explain how to bind data to data bind control
- ♦ explain how to display the data in data grid

6.1 INTRODUCTION

Most applications handle data, whether in the form of a database, text file, spreadsheet, whatever the majority of modern day programs need access to an information store.

Today, people are taking advantage of the latest data access technologies to improve scalability and boost interoperability.

ADO.NET advantages:

- **Interoperability** - All data in ADO.NET is transported in XML format, meaning it's simply a structured text document that can be read by anyone on any platform.
- **Scalability** - The client/server model is out. ADO.NET promotes the use of disconnected datasets, with automatic connection pooling bundled as part of the package.
- **Productivity** - You can't just pick it up and run, but ADO.NET can certainly improve your overall development time. For example, "Typed DataSets" help you work quicker and produce more bug-free code.
- **Performance** - Because ADO.NET is mainly about disconnected datasets, the database server is no longer a bottleneck and hence applications should incur a performance boost.

In ADO.NET, functionality is split into two key class groups — *content components* and *managed-provider components*. The content components essentially hold actual data and include the DataSet, DataTable, DataView, DataRow, DataColumn and DataRelation classes.

We also have the managed-provider components, which actually talk to the database to assist in data retrievals and updates. Such objects include the connection, command and data reader.

Also, the managed-provider components are split into two key groups — one designed for regular data sources, with another finely tuned specifically for SQL Server.

ADO.NET bundles with a bunch of content components. The most important are:

DataSet — This is a lot like the old Recordset object, except that it can hold multiple "tables" of data. You can also setup internal data constraints and relationships.

DataView — The DataView is similar to a regular database view. You can essentially use this object to filter tables inside the DataSet object.

Currently, there are two key sets of managed provider components — one designed for general data access (in System.Data.OleDb) and one fine-tuned for SQL Server (in System.Data.SqlClient).

Both of these comply with the standard data implementations defined in the System.Data.Common namespace.

So, what are the key managed-provider components?

Connection — *OleDbConnection* + *SqlConnection* — Like classic ADO, this object implements properties such as the connection string and state. We also have the typical .Open and .Close, plus .BeginTransaction returning an object to control a database transaction. Note that you no longer have an .Execute method on the Connection object.

Command — *OleDbCommand* + *SqlCommand* — This is the pipeline to the backend data. You can use the command to either .ExecuteNonQuery, which will action an SQL statement (such as a DELETE command) upon the data — or .ExecuteReader, which links straight in to the Data Reader object.

Data Reader — *OleDbDataReader* + *SqlDataReader* — This object essentially takes a stream of data from the Command object and allows you to read it. It's like a forward-only Recordset and is very efficient. However this uses a server-side cursor, so you should avoid it too much as it naturally requires an open connection.

Data Adapter — *OleDbDataAdapter* + *SqlDataAdapter* — The Data Adapter object provides an all-in-one stop for your data. It essentially serves as a middle man, going through your connection to retrieve data, then passing that into a DataSet. You can then pass the DataSet back to the Data Adapter, which will go and update the database. The SQL statements for each command are specified in the InsertCommand, UpdateCommand, and InsertCommand and DeleteCommand properties.

Sample access using Data Reader:

```
Setup connection
{
    System.Data.OleDb.OleDbConnection myConnection = new
System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" + "Data
Source=C:\\Program Files\\Microsoft Visual " +
"Studio.NET\\Common7\\Tools\\Bin\\nwind.Mdb");
    // Setup command object, specifying SELECT
    // and connection to use
    System.Data.OleDb.OleDbCommand myCommand = new
System.Data.OleDb.OleDbCommand("Select * from Customers", myConnection);
    System.Data.OleDb.OleDbDataReader myReader = null;
    // Open connection
    myConnection.Open();
    // Execute and put results into reader
    myReader = myCommand.ExecuteReader();
    // Read through all the records
    while (!(myReader.Read() == false)) {
        MessageBox.Show(myReader["CompanyName"]);
    }
    // Close reader connection before continuing
    myReader.Close();
    myConnection.Close();
}
```

```
}
```

Sample access using Data Adapter:

Setup connection

```
{
    System.Data.OleDb.OleDbConnection myConnection = new
System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" + "Data
Source=C:\\Program Files\\Microsoft Visual " +
"Studio.NET\\Common7\\Tools\\Bin\\nwind.Mdb");
    // We're just specifying the SELECT here.
    // If done visually using the Server Explorer,
    // the INSERT/UPDATE/DELETE + Parameters
    // collection would be auto-specified for us
    System.Data.OleDb.OleDbDataAdapter myDataAdapter = new
System.Data.OleDb.OleDbDataAdapter("Select * from Customers", myConnection);
    DataSet myDataSet = new DataSet();
    // Fill DataSet with table, call it 'Customers'
    myDataAdapter.Fill(myDataSet, "Customers");
    // Display first CompanyName field of first
    // row in Customers table
    MessageBox.Show(myDataSet.Tables["Customers"].Rows[0]["CompanyName
"]);
}
// If we had specified INSERT/UPDATE/DELETE
// commands + Parameters collection, we could
// also now edit the data, then run something like:
// myDataAdapter.Update(myDataSet)
```

Sample access using XML:

```
DataSet myDataSet = new DataSet();
myDataSet.ReadXml("c:\\books.xml");
// Books.xml is file bundled with VS.NET,
// typically located at:
// C:\\Program Files\\Microsoft.NET\\FrameworkSDK
// ... \\Samples\\quickstart\\howto\\samples\\xml
// ... \\xmldocumentevent\\vb\\books.xml
// Counts the number of book nodes
MessageBox.Show(myDataSet.Tables["book"].Rows.Count);
// Retrieves the last name of the first author
MessageBox.Show(myDataSet.Tables["author"].Rows[0]["last-name"]);
// Updates the last name
myDataSet.Tables["author"].Rows[0]["last-name"] = "Flandadenham";
// Rewrites the XML file
myDataSet.WriteXml("c:\\books.xml");
```

6.2 ADO .NET ARCHITECTURE

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access.

Evolution of ADO.NET

The first data access model, DAO (data access model) was created for local databases with the built-in Jet engine which had performance and functionality issues. Next data access model is RDO (Remote Data Object) and ADO (Active Data Object)

which were designed for Client Server architectures but soon ADO took over RDO. ADO was a good architecture but as the language changes so is the technology. With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls. ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed. Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic. Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.

Why ADO.NET?

ADO.NET addresses the above mentioned problems by maintaining a disconnected database access model which means, when an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is updated, the connection is opened long enough to complete the Update operation and is closed. By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO.NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

The ADO.NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

1) DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

2) Data Provider

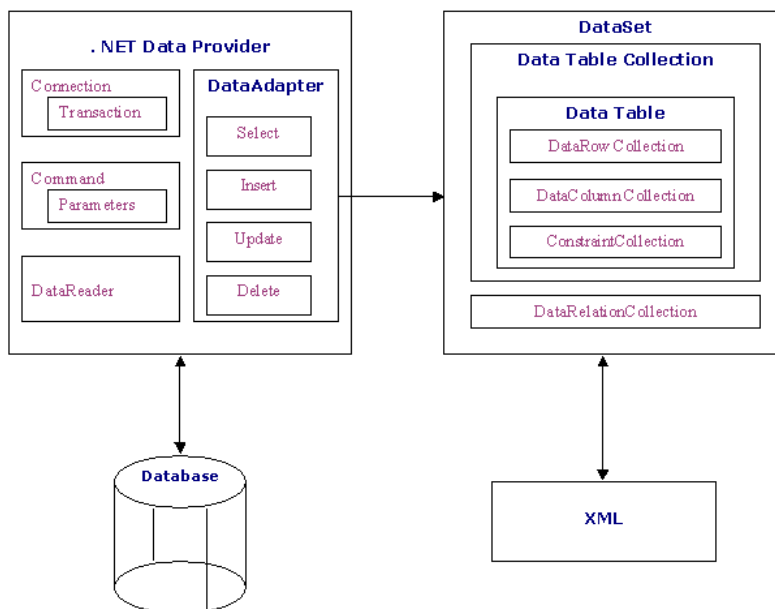
The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

The Connection object which provides a connection to the database
The Command object which is used to execute a command
The DataReader object which provides a forward-only, read only, connected recordset

The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows :

A connection object establishes the connection for the application with the database. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.



ADO .NET Data Architecture

Component classes that make up the Data Providers

The Connection Object

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the `SqlConnection` object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the `OleDbConnection` object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

The Command object is represented by two corresponding classes: `SqlCommand` and `OleDbCommand`. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:

ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE

ExecuteScalar: Returns a single value from a database query

ExecuteReader: Returns a result set by way of a `DataReader` object

The DataReader Object

The `DataReader` object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, `DataReader` objects cannot be directly instantiated. Rather, the `DataReader` is returned as the result of the Command object's `ExecuteReader` method. The `SqlCommand.ExecuteReader` method returns a `SqlDataReader` object, and the `OleDbCommand.ExecuteReader` method returns an `OleDbDataReader` object. The `DataReader` can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the `DataReader` provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the `DataReader`.

The DataAdapter Object

The `DataAdapter` is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a `DataSet`. The `DataAdapter` is used either to fill a `DataTable` or `DataSet` with data from the database with its `Fill` method. After the memory-resident data has been manipulated, the `DataAdapter` can commit the changes to the database by calling the `Update` method. The `DataAdapter` provides four properties that represent database commands:

Select Command
InsertCommand
DeleteCommand
UpdateCommand

When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

Check your progress 6.2

Answer the following questions in 1-2 sentences.

a) What is ADO.NET?

.....
.....

b) What is dataset?

.....
.....

6.3 ACCESSING DATA USING DATA ADAPTERS AND DATASETS

ASP.NET includes features that enable you to add data access to your ASP.NET Web pages with little or no code. You can connect to databases, XML data and files, and business objects as data sources. You can then display data by using a variety of controls that provide great flexibility in how you present data on the page.

What are DataSets and Data Adapters?

Datasets store a copy of data from the database tables. However, Datasets cannot directly retrieve data from Databases. DataAdapters are used to link Databases with DataSets. If we see diagrammatically,

DataSets < ----- DataAdapters < ----- DataProviders < ----- Databases

DataSets and DataAdapters are used to display and manipulate data from databases.

Reading Data into a Dataset

To read data into Dataset, you need to:

Create a database connection and then a dataset object.

Create a DataAdapter object and refer it to the DB connection already created. Note that every DataAdapter has to refer to a connection object. For example, SqlDataAdapter refers to SqlConnection.

The *Fill* method of DataAdapter has to be called to populate the Dataset object.

We elaborate the above mentioned steps by giving examples of how each step can be performed:

1) As we said, our first task is to create a connection to database. We would explore later that there is no need of opening and closing database connection explicitly while you deal with DataAdapter objects. All you have to do is, create a connection to database using the code like this:

```
SqlConnection con = new SqlConnection ("data source=localhost; uid= sa; pwd= abc; database=Northwind");
```

We would use Northwind database by using OleDbConnection.

The Code would Look like:

```
OleDbConnection con= new OleDbConnection ("Provider =Microsoft.JET.OLEDB.4.0;" + "Data Source=C:\\Program Files\\Microsoft Office\\Office\\Samples\\Northwind.mdb");
```

2) Now, create a Dataset object which would be used for storing and manipulating data. You would be writing something like

```
DataSet myDataSet = new DataSet ("Northwind");
```

Since the name of source database is Northwind, we have passed the same name in the constructor.

3) The DataSet has been created but as we said before, this DataSet object cannot directly interact with Database. We need to create a DataAdapter object which would refer to the connection already created. The following line would declare a DataAdapter object:

```
OleDbAdapter myDataAdapter = new OleDbAdapter (CommandObject, con);
```

The above line demonstrates one of many constructors of OleDbAdapter class. This constructor takes a command object and a database connection object. The purpose of command object is to retrieve suitable data needed for populating DataSet. As we know SQL commands directly interacting with database tables, a similar command can be assigned to CommandObject.

```
OleDbCommand CommandObject = new OleDbCommand ("Select * from employee");
```

Whatever data you need for your Dataset should be retrieved by using suitable command here. The second argument of OleDbAdapter constructor is connection object *con*.

Alternative approach for initializing DataAdapter object:

Place a null instead of CommandObject while you initialize the OleDbAdapter object:

```
OleDbAdapter myDataAdapter = new OleDbAdapter (null, con);
```

Then you assign your query to the CommandObject and write:

```
myDataAdapter.SelectCommand = CommandObject;
```

4) Now, the bridge between the DataSet and Database has been created. You can populate dataset by using the *Fill* command:

```
myDataAdapter.Fill (myDataSet, "EmployeeData");
```

The first argument to *Fill* function is the DataSet name which we want to populate. The second argument is the name of *Data Table*. The results of SQL queries go into DataTable. In this example, we have created a DataTable named EmployeeData and the values in this table would be the results of SQL query: "Select * from employee". In this way, we can use a dataset for storing data from many database tables.

5) DataTables within a Dataset can be accessed using *Tables*. To access EmployeeData, we need to write:

```
myDataSet.Tables["EmployeeData"].
```

To access rows in each Data Table, you need to write:

```
myDataSet.Tables["EmployeeData"].Rows
```

Listing 1.1 would combine all the steps we have elaborated so far.

```
<%@ Page Language= "C#" %>
<%@ Import Namespace= "System.Data" %>
<%@ Import Namespace= "System.Data.OleDb" %>
<html>
<body>

<table border=2>
<tr>
<td><b> Employee ID </b></td>
<td><b> Employee Name </b></td>
</tr>

<% OleDbConnection con= new OleDbConnection ("Provider
=Microsoft.JET.OLEDB.4.0;" + "Data Source=C:\\Program Files\\Microsoft
Office\\Office\\Samples\\Northwind.mdb");
<%
DataSet myDataSet = new DataSet();
```

```

OleDbCommand CommandObject = new OleDbCommand ("Select * from
employee");

OleDbAdapter myDataAdapter = new OleDbAdapter (CommandObject, con);
myDataAdapter.Fill (myDataSet, "EmployeeData");

foreach(DataRow dr in myDataSet.Tables["EmployeeData"].Rows)
{
    Response.write ("<tr>");
    for (int j = 0; j <2 ; j++)
    {
        Response.write ( "<td>" + dr[j].ToString() + "</td>" );
    }
    Response.write ("</tr>");

    %>
</table>
</body>
</html>

```

The Code above would iterate in all rows of Employee table and display ID and name of every employee.

```
for (int j = 0 ; j < dr.Table.Columns.Count ; j++)
```

As we said earlier, there is no need of opening and closing database connection explicitly. DataAdapter class handles both these functions.

6.4 BINDING DATA TO DATA BIND CONTROLS

Over the last few chapters we've introduced many of the Web Controls you use when creating dynamic web pages. The "data" we used was somehow static. For example, to populate a ListBox, we've added the items from code by using a predefined set of values. Web Forms, as Windows forms do, allow displaying information by binding controls to a source of data - whatever that source of data is (database data, XML data, some array of values, etc.).

Throughout this chapter, you'll learn about the following:

- ☐ What does data binding stand for in ASP.NET applications
- ☐ The nature of "stateless" data binding
- ☐ The DataBind method, and binding Web Controls for selecting choices

Data Binding

Since Web Forms are stateless (all the information displayed on a Web Form gets **lost** between postbacks), data binding in Web Forms is somehow different from binding data in traditional Windows applications. We've all used to, for example, connect a *TDBGrid* control to a *TDataSource* control to display (and edit) data from some *TDataSet* control - in (Windows) Delphi database applications.

Although data binding, as a term, is often related to displaying data from a database (using ADO.NET and/or BDP.NET), we'll first explore binding Web control using data *not* coming from some database. This way we'll focus on the core issues with ASP.NET data binding - working with databases will be covered in some future chapters.

The Nature of ASP.NET data binding

In Web Form applications, most of the data displayed to the user (visitor of the Web Site), is provided as read-only. For example, a search result page, in some online store, will display information about the searched products with optional links to a page with product details. The (search results) data can be retrieved from a database, with no need to enable users to enter data to be saved back to the database.

With the above discussion in mind, here are some of the notes on Web Forms data binding:

- ASP.NET data-binding architecture does not perform (automatic) updates - Web forms data binding is always **read-only**.
- ASP.NET data binding is **not** limited to using ADO.NET (database) sources.
- Data binding often requires using *binding expressions* directly in the aspx file. We'll discuss binding expressions in the next chapter.
- While data binding provides control over what data is used for binding, when it is used and in what format, ASP.NET has the final word.
- Data binding can be used to assign data values to a specific property of a control. Use the *DataBindings* entry in the Object Inspector at design time; use the *DataBinding* event at run time to data bind single-value controls at run time (again, this will be discussed in the coming chapters).
- Multi-record controls (ListBox, DataList, Repeater, etc.) are bound to some DataSource to provide data for multiple items.

The DataBind method and Data Bound Lists

The simplest form of ASP.NET data binding is the automatic population of web controls, such as the ListBox, RadioButtonList, CheckBoxList, etc. Each of these controls inherits from a base *ListControl* class. The ListControl class defines a set of properties (and methods) used when doing data binding.

What's more, any Web Control can be bound to some data source - since the overall ancestor of every Web Control is the *Control* class, and the Control class has the DataBind method. Data binding is in the heart of ASP.NET development.

Note: the ListControl has a property named *Items*. This is a collection of *ListItem* objects. The ListItem class has the *Text* and *Value* properties. Text is the string value displayed in a list control, Value is an arbitrary (string) value associated with the item.

Of course, we've already used ListItem objects. Here's a line of code to add an item to a ListBox control (named ListBox1): `ListBox1.Items.Add(ListItem.Create('Text', 'Value'));`

Here's the set of common members of the ListControl class, related to data binding:

- The **DataSource property** defines the source of data for the control. While DataSource is of the System.Object (TObject) type, an object must implement either the IEnumerable or IList interface to be used as the source of data for binding.
When a control is "armed" with the IEnumerable interface, it provides iteration over a collection. Many of the classes in the .NET framework implement the IEnumerable interface: Array, ArrayList, Hashtable, DataReader, DataView, etc. Many of these classes implement the IList interface, too. The IList interface allows for a collection to be accessed by an index. With the above in mind, it is quite clear that data binding in ASP.NET is all but NOT strictly related to retrieving the information from some kind of database. A simple array of string values can be used as the DataSource.
- The **DataTextField property** specifies the field (property) of the DataSource object to be used for the visible content of the control (the Text property of the underlying ListItem object).
- The **DataValueField property** specifies the field (property) of the DataSource object to be used for the value content of the control - The Value property of the ListItem.
- The **DataTextFormatting property** enables specifying the .NET format string to be used when displaying the value for the DataTextField.

- And finally, the **DataBind method**. DataBind is called to activate the binding after, at least, the DataSource has been provided.

When DataBind is executed for a control, it will call DataBind for all its child controls. What this means is that if you have several controls on a Web Forms page, you can call Page.DataBind for the Page! This will call DataBind for all the controls that have their DataSource property set.

6.5 DISPLAYING DATA IN DATA GRID

The DataGrid control displays the fields of a data source as columns in a table. Each row in the control represents a record in the data source. The control supports selection, editing, deleting, paging, and sorting.

The DataGrid control with strong features is the most complicated control included within the ASP.NET framework. Like the Repeater and DataList controls, it enables to format and display records from a database table. However, it has several advanced features, such as support for sorting and paging through records, which makes it unique.

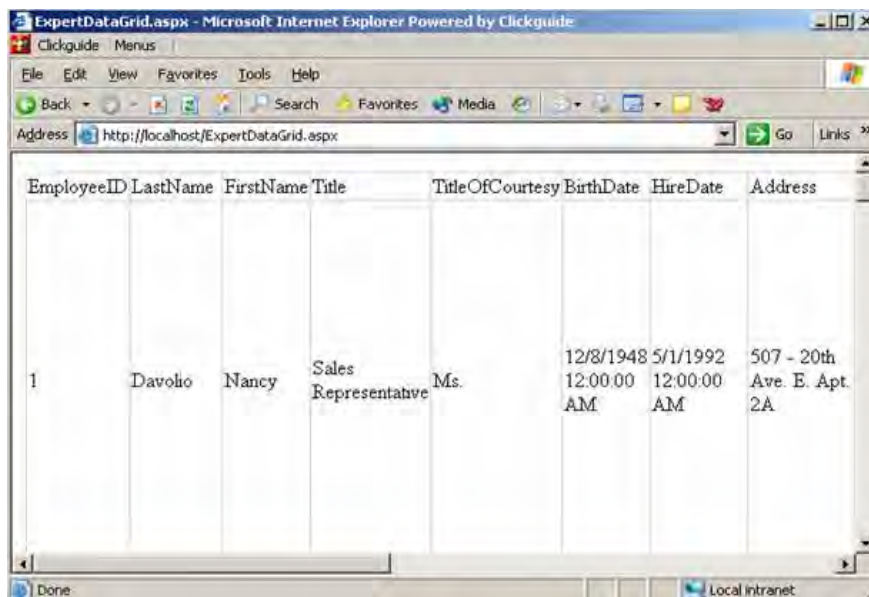
Records can be displayed in a DataGrid without using templates. A data source can be simply bound to the DataGrid, and it automatically displays the records.

The following example, displays all the records from the Employees database table in a DataGrid

Example : ExpertDataGrid.aspx

```
<%@ Import Namespace="System.Data.SqlClient" %>
<Script language = "VB" Runat="Server">
Sub Page_Load
Dim conNorthwind As SqlConnection
Dim cmdSelect As SqlCommand
conNorthwind=New SqlConnection(
"Server=localhost;UID=sa;PWD=secret;Database=Northwind" )
cmdSelect = New SqlCommand( "Select * From Employees", conNorthwind )
conNorthwind.Open()
dgrdEmployees.DataSource = cmdSelect.ExecuteReader()
dgrdEmployees.DataBind()
conNorthwind.Close()
End Sub
</Script>
<html>
<head><title>ExpertDataGrid.aspx</title></head>
<body>
<asp:DataGrid
ID="dgrdEmployees"
Runat="Server" />
</body>
</html>
```

The output of above example is shown below:



EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate	Address
1	Davolio	Nancy	Sales Representative	Ms.	12/8/1948 12:00:00 AM	5/1/1992 12:00:00 AM	507 - 20th Ave. E. Apt. 2A

By default, a DataGrid displays gridlines around its items. Modification of the Grid line appearance can be done by setting the GridLines property. The possible values are Both, Horizontal, None, or Vertical.

For example, to completely disable GridLines, the DataGrid would look like this: -

```
<asp:DataGrid  
GridLines="None"  
Runat="Server" />
```

The cell spacing and cell padding of the cells in a DataGrid can be controlled by modifying the DataGrid control's CellSpacing and CellPadding properties like this:

```
<asp:DataGrid  
CellSpacing="10"  
CellPadding="10"  
Runat="Server">
```

A background image can be specified for a DataGrid by assigning the name of an image to the BackImageUrl property.

For example, the following DataGrid displays an image named Bricks. Gif in the background:

```
<asp:DataGrid  
BackImageUrl="expert.Gif"  
Runat="Server" />
```

Finally, headers and footers can be displayed and hidden for the columns in a DataGrid by enabling or disabling the ShowHeader and ShowFooter properties. By default the ShowHeader property has the value True, and the ShowFooter property has the value False. To prevent column headers from being displayed, a DataGrid would be like this:

```
<asp:DataGrid  
ShowHeader="False"  
Runat="Server" />
```

Creating Columns in a DataGrid Control

The DataGrid control displays the columns in a variety of ways. By default, the columns are generated automatically based on fields in the data source. However, in order to control the content and layout of columns more precisely, the following types of columns can be defined:

Type of Column	Description
Bound column	Allows specifying which data source field to display and specifies the format of that field, using a .NET formatting expression. For details see Adding Bound Columns to a DataGrid Web Server Control.
Hyperlink column	Displays information as hyperlinks. A typical use is to display data (such as a customer number or product name) as a hyperlink that users can click to navigate to a separate page that provides details about that item. For details see Adding Hyperlink Columns to a DataGrid Web Server Control.
Button column	Allows adding a button for each item in the grid and defining custom functionality for that button. For example, you might create a button labeled "Add to Shopping Cart" that runs your custom logic when a user clicks it. You can also add predefined buttons for Select, Edit, Update, Cancel, and Delete functions.
Edit, Update, Cancel column	Allows creating in-place editing. For more details, see "Editing Items" below.
Template column	Allows creating combinations of HTML text and server controls to design a custom layout for a column. The controls within a template column can be data-bound. Template columns gives great flexibility in defining the layout and functionality of the grid contents, because you have complete control over how the data is displayed and what happens when users interact with rows in the grid. For details see Adding Template Columns to a DataGrid Web Server Control.

Data Grid Events

The DataGrid control supports several events.

One of them, the ItemCreated event, gives you a way to customize the item-creation process. The ItemDataBound event also gives you the ability to customize the DataGrid items, but after the data is available for inspection. For example, if you were using the DataGrid control to display a to-do list, you could display overdue items in red text, completed items in black text, and other tasks in green text.

The remaining events are raised in response to button or LinkButton clicked in grid items. They are designed to implement common data manipulation tasks. Four events of this type are supported:

EditCommand
DeleteCommand
UpdateCommand
CancelCommand

When the user clicks one of the buttons (labeled by default Edit, Delete, Update, or Cancel, respectively), the corresponding event is raised.

The DataGrid control also supports the ItemCommand event that is raised when a user clicks a button that is not one of the predefined buttons above. This event can be used for custom functions by setting a button's CommandName property to a value needed, and then testing for it in the ItemCommand event handler.

(For example, you could use this approach when selecting an item, as documented in allowing Users to Select Items in a DataList Web Server Control.) By default, a DataGrid simply displays all the columns from its data source. However, if False value is assigned to the DataGrid control's **AutoGenerateColumns** property, columns can be created individually to have more control over the formatting.

Adding a BoundColumn to a DataGrid

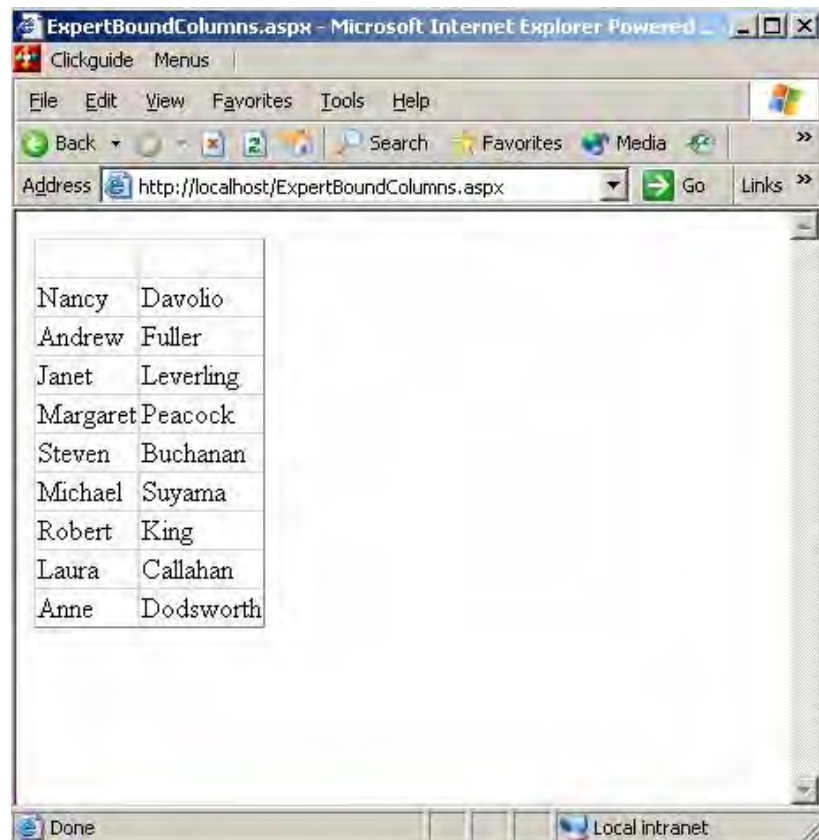
The default column used in a DataGrid is a BoundColumn. If only limited columns are to be displayed and controlled from a data source, declaration of one or

more BoundColumns controls is done explicitly. The following example demonstrates it :-

Example : ExpertBoundColumns.aspx

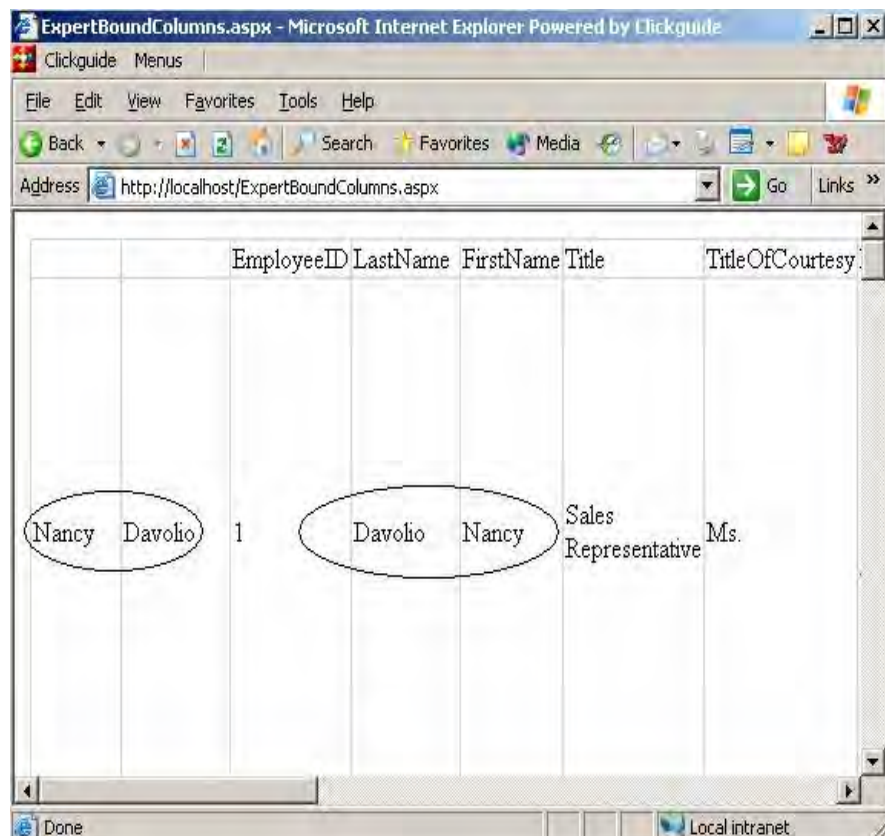
```
<%@ Import Namespace="System.Data.SqlClient" %>
<Script language = "VB" Runat="Server">
Sub Page_Load
Dim conNorthwind As SqlConnection
Dim cmdSelect As SqlCommand
conNorthwind=New SqlConnection(
"Server=localhost;UID=sa;PWD=secret;Database=Northwind" )
cmdSelect = New SqlCommand( "Select * From Employees", conNorthwind )
conNorthwind.Open()
dgrdEmployees.DataSource = cmdSelect.ExecuteReader()
dgrdEmployees.DataBind()
conNorthwind.Close()
End Sub
</Script>
<html>
<head><title>ExpertBoundColumns.aspx</title></head>
<body>
<asp:DataGrid
ID="dgrdEmployees"
AutoGenerateColumns="False"
EnableViewState="False"
Runat="Server">
<Columns>
<asp:BoundColumn DataField="FirstName" />
<asp:BoundColumn DataField="LastName" />
</Columns>
</asp:DataGrid>
</body>
</html>
```

The output of above example is shown below:



The above example, displays only the FirstName and LastName columns from the Employees table.

Note that the DataGrid control's AutoGenerateColumns property is set to the value False so that, all the columns from the Employees table are displayed, and the FirstName and LastName columns are displayed twice as shown below: -



Therefore it is important to take care of AutoGenerateColumns property of DataGrid control.

Further in the above example the BoundColumns are declared inside the DataGrid control's <Columns> tag. This tag contains a BoundColumn for the Titles column and a BoundColumn for the Price column. The DataField property indicates the field from the data source to display in the BoundColumn.

Adding a HyperLinkColumn to DataGrid Control

This is a very important concept in DataGrid Controls. To display links to other pages from the DataGrid, a HyperLinkColumn can be used.

This type of column is useful in two situations.

First, a HyperLinkColumn can be used, for example create a master-detail form. A set of master records can be displayed in a DataGrid and a HyperLinkColumn can be used in the DataGrid to link to a separate page that contains detailed information.

Secondly it can be used for interlinking pages and websites with each other, an important need of web programmers.

For example, you can use a HyperLinkColumn to display a list of useful ASP.NET Web sites.

Example : ExpertDataGridHyperLink.aspx

```
<%@ Import Namespace="System.Data.SqlClient" %>
<Script language = "VB" Runat="Server">
  Sub Page_Load
    Dim conNorthwind As SqlConnection
    Dim cmdSelect As SqlCommand
    conNorthwind=NewSqlConnection(
    "Server=localhost;UID=sa;PWD=secret;Database=Northwind" )
    cmdSelect = New SqlCommand( "Select * From Employees", conNorthwind )
    conNorthwind.Open()
    dgrdEmployees.DataSource = cmdSelect.ExecuteReader()
    dgrdEmployees.DataBind()
    conNorthwind.Close()
  End Sub
</Script>
<html>
<head><title>ExpertDataGridHyperLink.aspx</title></head>
<body>
<form Runat="Server">
  <asp:DataGrid
    ID="dgrdEmployees"
    AutoGenerateColumns="False"
    EnableViewState="False"
    CellPadding="10"
    Runat="Server">
    <Columns>
    <asp:BoundColumn
      HeaderText="Employee"
      DataField="FirstName" />
    <asp:HyperLinkColumn
      HeaderText="Complete Details"
      DataNavigateUrlField="EmployeeID"
      DataNavigateUrlFormatString="EmpDetails.aspx?id={0}"
      Text="view complete details" />
    </Columns>
  </asp:DataGrid>
</form>
</body>
</html>
```

The output of the above example is shown below:



The DataGrid in above example is bound to the Employees database table.

The values from the Employees FirstName column are displayed by the BoundColumn, and links to a page with detailed author information are displayed by the HyperLinkColumn. Each hyperlink displayed by the HyperLinkColumn is displayed with the help of the DataNavigateUrlField, DataNavigateUrlFormatString, and Text properties. The DataNavigateUrlField and DataNavigateUrlFormatString properties create the URL for the link. In this case, the links point to a page named EmpDetails.aspx. The EmployeeID column is passed to EmpDetails.aspx in a query string variable named ID.

The Text property specifies the label for the link. In above example, each link is simply labeled with the text view of complete details.

Check your progress 6.3 – 6.5

Answer the following questions in 1-2 sentences.

a) What is data adapter?

.....
.....

b) How to display the data in data grid control?

.....
.....

6.6 SUMMARY

- DataSet — this is not like the old Recordset object, except that it can hold multiple "tables" of data. You can also setup internal data constraints and relationships.
- DataView — the DataView is similar to a regular database view. You can essentially use this object to filter tables inside the DataSet object.
- The Connection object which provides a connection to the database.
- The Command object which is used to execute a command. The DataReader object which provides a forward-only, read only, connected recordset
- The DataAdapter object which populates a disconnected DataSet with data and performs update
- The DataGrid control displays the fields of a data source as columns in a table. Each row in the control represents a record in the data source. The control supports selection, editing, deleting, paging, and sorting.

6.7 CHECK YOUR PROGRESS – ANSWERS

6.2

1. ADO .NET is a connecting layer between your application and database. It is introduced for sake of flexibility for connectivity.
2. Dataset is a temporary place for containing data / table / fields for use in your application.

6.3-6.5

1. Data adapter is concerned to fill your data to your application, because it cannot extract data from your d/b. so it is in mid of your application and dataset.
2. Data grid displays all the records in column format, using data source. We can arrange data in datagrid in any order i.e. ascending/descending.

6.8 QUESTIONS FOR SELF - STUDY

1. Explain the advantages of ADO.NET.
2. Explain ADO.NET data architecture.
3. Explain data adapter and dataset.
4. Explain data grid with example.

6.9 SUGGESTED READINGS

1. Programming C#: Building .NET Applications with C# By Jesse Liberty
2. Learning C# 2005 by Jesse Liberty
3. Microsoft MSDN
4. Beginning ASP.NET 4.5 in C#, By Matthew MacDonald
5. <http://delphi.about.com/library/weekly/aa012505a.htm>



NOTES

[illegible]