S.Y.B.Sc. (C. S.)

SEMESTER - IV (CBCS)

.NET TECHNOLOGIES

SUBJECT CODE: USCS406

Prof. (Dr.) D. T. Shirke
Offg. Vice Chancellor
University of Mumbai, Mumbai

Prin. Dr. Ajay Bhamare
Offg. Pro Vice-Chancellor,
University of Mumbai

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology IDOL,
Univeristy of Mumbai – 400098

Course Co-ordinator : **Ms. Mitali Vijay Shewale**
Doctoral Researcher,
Veermata Jijabai Technological Institute
HR Mahajani road, Matunga, Mumbai

Editor : **Dr Rajeshri Shinkar**
Assistant Professor,
IES College, Nerul, Navi Mumbai

Course Writers : **Dr. Shraddha Bhushan Sable**
Assistant Professor,
S. K. College of Sci & Commerce,
Nerul, Navi Mumbai,

: **Ms. Jyoti Uday Darne**
Assistant Professor,
Dr. Pillai Global Academy, New Panvel.

: **Mr. Milind Thorat**
Lecturer,
K. J. Somaiya Institute of Engineering and
Information Technology, Sion East, Mumbai.

June 2023, Print - 1

Published by : Director,
Institute of Distance and Open Learning,
University of Mumbai,
Vidyanagari,Mumbai - 400 098.

DTP composed and Printed by: Mumbai University Press

CONTENTS

S.Y.B.Sc. (C. S.)

SEMESTER - IV (CBCS)

.NET TECHNOLOGIES

SYLLABUS

Course: **USCS406**	**TOPICS (Credits : 02 Lectures/Week: 03)** **.Net Technologies**	
Objectives: To explore .NET technologies for designing and developing dynamic, interactive and responsive web applications. **Expected Learning Outcomes:** 1. Understand the .NET framework 2. Develop a proficiency in the C# programming language 3. Proficiently develop ASP.NET web applications using C# 4. Use ADO.NET for data persistence in a web application		
Unit I	**The .NET Framework:**.NET Languages, Common Language Runtime, .NET Class Library **C# Language Basics:** Comments, Variables and Data Types, Variable Operations, Object-Based Manipulation, Conditional Logic, Loops, Methods, Classes, Value Types and Reference Types, Namespaces and Assemblies, Inheritance, Static Members, Casting Objects, Partial Classes **ASP.NET:** Creating Websites, Anatomy of a Web Form - Page Directive, Doctype, Writing Code - Code-Behind Class, Adding Event Handlers, Anatomy of an ASP.NET Application - ASP.NET File Types, ASP.NET Web Folders, **HTML Server Controls** - View State, HTML Control Classes, HTML Control Events, HtmlControl Base Class, HtmlContainerControl Class, HtmlInputControl Class, Page Class, global.asax File, web.config File	**15L**
Unit II	**Web Controls:** Web Control Classes, WebControl Base Class, List Controls, Table Controls, Web Control Events and AutoPostBack, Page Life Cycle **State Management:** ViewState, Cross-Page Posting, Query String, Cookies, Session State, Configuring Session State, Application State **Validation:** Validation Controls, Server-Side Validation, Client-Side Validation, HTML5 Validation, Manual Validation, Validation with Regular Expressions **Rich Controls:** Calendar Control, AdRotator Control, MultiView Control **Themes and Master Pages:** How Themes Work, Applying a Simple Theme,	**15L**

	Handling Theme Conflicts, Simple Master Page and Content Page, Connecting Master pages and Content Pages, Master Page with Multiple Content Regions, Master Pages and Relative Paths **Website Navigation:** Site Maps, URL Mapping and Routing, SiteMapPath Control, TreeView Control, Menu Control	
Unit III	**ADO.NET:** Data Provider Model, Direct Data Access - Creating a Connection, Select Command, DataReader, Disconnected Data Access **Data Binding**: Introduction, Single-Value Data Binding, Repeated-Value Data Binding, Data Source Controls – SqlDataSource **Data Controls:** GridView, DetailsView, FormView **Working with XML:** XML Classes – XMLTextWriter, XMLTextReader **Caching:** When to Use Caching, Output Caching, Data Caching **LINQ:** Understanding LINQ, LINQ Basics, **ASP.NET AJAX:** ScriptManager, Partial Refreshes, Progress Notification, Timed Refreshes	**15L**

Textbook(s):

1) Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)

Additional Reference(s):

1) The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
2) Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

1

THE .NET FRAMEWORK

Unit Structure :

1.0 INTRODUCTION

The .NET Framework

.NET framework is an integral windows component that helps in building and executing the next generation of applications and XML web services. It is a set of Microsoft software technologies for connecting your world of information, people, systems, and devices.

1.1 OBJECTIVES OF .NET FRAMEWORK

- To provide a very high degree of language interoperability
- To provide a runtime environment that completely manages code execution
- To provide high-level code security through code access security and strong type checking
- To facilitate application communication by using industry standards such as SOAP and XML.
- To simplify Web application development
- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.

- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

1.2 COMPONENTS OF .NET FRAMEWORK

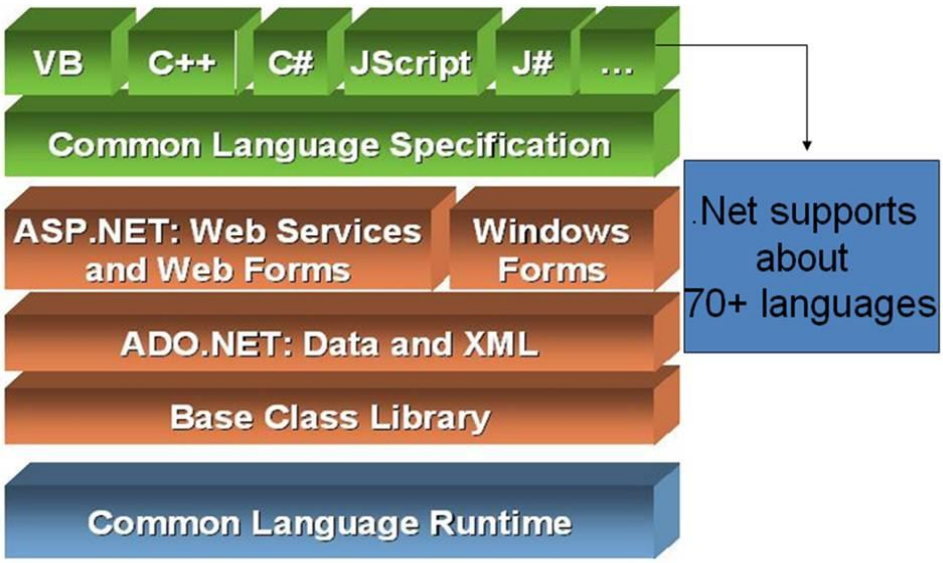


Fig. 1.1 Components of .NET framework

- The .Net framework allows infrastructural services to all the applications developed in .Net compliant language.
- It is an engine that provides runtime services using its component like Common Runtime Language.
- The .Net framework provides tools and technologies to develop windows and web applications.
- The .Net framework mainly contains two components :

  1. Common Language Runtime(CLR)
  2. .Net Framework Class Library (FCL)

1. Common Language Runtime (CLR)

- .Net Framework provides runtime environment called Common Language Runtime (CLR).

- It runs all the .Net programs.
- CLR provides memory management and thread management.
- It allocates the memory for scope and deallocates the memory.
- The code which runs under the CLR is called as Managed Code.
- Programmers need not to worry on managing the memory if the programs are running under the CLR. (memory management and thread management)
- Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to Microsoft Intermediate Language(MSIL) intern this will be converted to Native Code by CLR.
- There are currently over 15 language compilers being built by Microsoft and other companies also producing the code that will execute under CLR.

2. .Net Framework Class Library (FCL)

- It accesses the library classes and methods.
- It is also called as Base Class Library.
- It is common for all types of application.

  Following are the applications in .Net Class Library:

  1. XML web services
  2. Windows services
  3. Windows application
  4. Web applications
  5. Console application

1.3 .NET FRAMEWORK DESIGN PRINCIPLE

Interoperability – The .Net framework provides a lot of backward support. Suppose you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine that had the higher version of the .Net framework, say 3.5. The application would still work. This is because Microsoft ensures that older framework versions gel well with the latest version.

Portability – Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning making Microsoft products work on other platforms, such as iOS and Linux.

Security – The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both the validation and verification of

applications. Every application can explicitly define its security mechanisms. Each security mechanism is used to grant the user access to the code or to the running program.

Memory management – The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the "Garbage Collector" which runs as part of the .Net framework. The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

Simplified deployment – The .Net framework also has tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

1.4 .NET LANGUAGES

.NET Languages are computer programming languages that are used to produce programs that execute within the Microsoft .NET Framework. Microsoft provides several such languages, including C#, Visual Basic .NET, and C++/CLI. Regardless of which .NET language is used, the output of the language compiler is a representation of the same logic in an intermediate language named **C**ommon **I**ntermediate **L**anguage. Before the program is executed, CIL is compiled to object code appropriate for the machine on which the program is executing. This last compilation step is usually performed by the Common Language Runtime component of the framework at the moment the program is invoked, though it can be manually performed at an earlier stage.

While there are currently more than 40 languages with compilers for the .NET Framework, only a small number of them are widely used and supported by Microsoft. The rest is composed of languages developed by third party vendors.

The types of applications that can be built in the .Net framework are classified broadly into the following categories.

WinForms – This is used for developing Forms-based applications, which would run on an end-user machine. Notepad is an example of a client-based application.

ASP.Net – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome, or Firefox. The Web application would be processed on a server, which would have Internet Information Services Installed. Internet Information Services or IIS is a Microsoft component that is used to execute an Asp.Net application. The result of the execution is then sent to the client machines, and the output is shown in the browser.

ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

Few examples of Microsoft .NET languages

C# - Microsoft's flagship .NET Framework language which bears similarities to the C++ and Java languages.

Visual Basic .NET - A completely redesigned version of the Visual Basic language for the .NET Framework. This also includes Visual Basic 2005 (v8.0).

VBx, a dynamic version of Visual Basic .NET that runs on top of the Dynamic Language Runtime.

C++/CLI and the deprecated Managed C++ - A managed version of the C++ language.

J# - A Java and J++ .NET transitional language.

JScript .NET - A compiled version of the JScript language.

Windows PowerShell - An interactive command line shell/scripting language that provides full access to the .NET Framework.

IronPython - A .NET implementation of the Python programming language developed by Jim Hugunin at Microsoft.

IronRuby - A dynamically compiled version of the Ruby programming language targeting the .NET Framework.

F#, a member of the ML programming language family.

1.5 COMMON LANGUAGE RUNTIME (CLR)

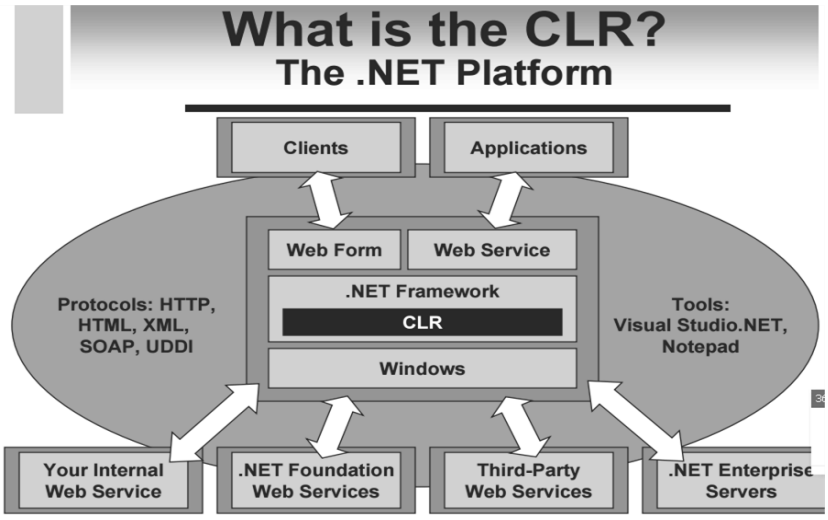


fig 1.2 CLR

As part of Microsoft's .NET Framework, the Common Language Runtime (CLR) is programming that manages the execution of programs written in any of several supported languages, allowing them to share common object-oriented classes written in any of the languages.

Benefits of CLR :

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded, scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.
- Use of delegates(a class that can hold a reference to a method) instead of function pointers for increased type safety and security.

.NET CLR is a runtime environment that manages and executes the code written in any .NET programming language. CLR is the virtual machine component of the .NET framework. Language's compiler compiles the source code of applications developed using .NET compliant languages into CLR's intermediate language called MSIL, i.e., Microsoft intermediate language code. This code is platform-independent. It is comparable to byte code in java. Metadata is also generated during compilation and MSIL code and stored in a file known as the Manifest file. This metadata is generally about members and types required by CLR to execute MSIL code. A just-in-time compiler component of CLR converts MSIL code into the native code of the machine. This code is platform-dependent. CLR manages memory, threads, exceptions, code execution, code safety, verification, and compilation.

fig 1.3 conversion process of source code to native code

Main components of CLR

- Common type system
- Common language speciation

- Garbage Collector
- Just in Time Compiler
- Metadata and Assemblies

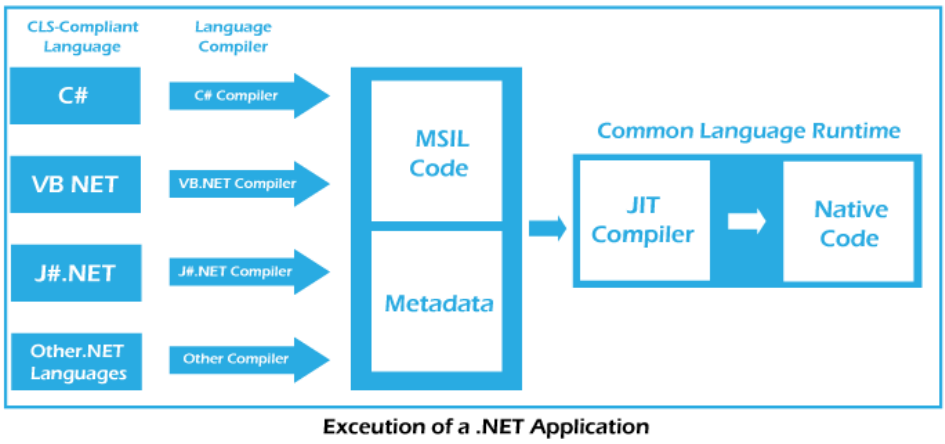


fig 1.4 .NET application processing

Common type system (CTS)

CTS provides guidelines for declaring, using and managing data types at runtime. It offers cross-language communication. For example, VB.NET has an integer data type, and C# has an int data type for managing integers. After compilation, Int32 is used by both data types. So, CTS provides the data types using managed code. A common type system helps in writing language-independent code.

Common Language Specification (CLS)

Common Language Specification (CLS) contains a set of rules to be followed by all NET-supported languages. The common rules make it easy to implement language integration and help in cross-language inheritance and debugging. Each language supported by NET Framework has its own syntax rules. But CLS ensures interoperability among applications developed using NET languages.

Garbage Collection

Garbage Collector is a component of CLR that works as an automatic memory manager. It helps manage memory by automatically allocating memory according to the requirement. It allocates heap memory to objects. When objects are not in use, it reclaims the memory allocated to them for future use. It also ensures the safety of objects by not allowing one object to use the content of another object.

Just in Time (JIT) Compiler

JIT Compiler is an important component of CLR. It converts the MSIL code into native code (i.e., machine-specific code). The .NET program is compiled either explicitly or implicitly. The developer or programmer calls

a particular compiler to compile the program in the explicit compilation. In implicit compilation, the program is compiled twice. The source code is compiled into Microsoft Intermediate Language (MSIL) during the first compilation process. The MSIL code is converted into native code in the second compilation process. This process is called JIT compilation.

Metadata

Metadata is binary information about the program, either stored in a CLR Portable Executable file (PE) along with MSIL code or in the memory. During the execution of MSIL, metadata is also loaded into memory for proper interpretation of classes and related. Information used in code. So, metadata helps implement code in a language-neutral manner or achieve language interoperability.

Assemblies

An assembly is a fundamental unit of physical code grouping. It consists of the assembly manifest, metadata, MSIL code, and a set of resources like image files. It is also considered a basic deployment unit, version control, reuse, security permissions, etc.

Functions of CLR

Following are the functions of the CLR.

- It converts the program into native code.
- Handles Exceptions
- Provides type-safety
- Memory management
- Provides security
- Improved performance
- Language independent
- Platform independent
- Garbage collection
- Provides language features such as inheritance, interfaces, and overloading for object-oriented programs.

1.6 .NET CLASS LIBRARY

.NET Framework Class Library is the collection of classes, namespaces, interfaces, and value types that are used for .NET applications.

It contains thousands of classes that support the following functions.

- Base and user-defined data types
- Support for exceptions handling
- input/output and stream operations

- Communications with the underlying system
- Access to data
- Ability to create Windows-based GUI applications
- Ability to create web client and server applications
- Support for creating web services

The .NET Framework provides a large and very rich library of classes to be used and extended by application developers. Reuse and extension of these classes will allow developers to be more productive and to develop more robust and feature-rich applications in a shorter time frame because the class library provides many features that previously had to be built from scratch.

.NET class library is divided into namespaces so that it can be easy to work with and understand. The System namespace is considered the root namespace which acts as a container for all the base data type classes used by application developers to build frameworks and applications. The .NET class library is common to all languages of .NET. In other words, the way one access files in C# will be exactly the same in VB.NET and for all other languages of .NET.

The .NET Framework class library contains classes that allow the development of the following type of applications:

- Console applications
- Window applications
- Windows services
- ASP .NET Web applications
- Web services
- Windows communication foundation applications
- Windows presentation foundation applications
- Window workflow foundation applications

The library's classes are organized using a hierarchy of namespace. For example, all the classes performing I/O operations are located in the System.IO namespaces, and classes that manipulate regular expressions are located in the System.Text.RegularExpressions namespace.

Class libraries are the shared library concept for .NET. They enable you to componentize useful functionality into modules that can be used by multiple applications. They can also be used as a means of loading functionality that is not needed or not known at application startup. Class libraries are described using the .NET Assembly file format.

There are three types of class libraries:-

Platform-specific class libraries: They have access to all the APIs in a given platform (for example, .NET Framework on Windows, Xamarin iOS), but can only be used by apps and libraries that target that platform.

Portable class libraries: They have access to a subset of APIs, and can be used by apps and libraries that target multiple platforms.

.NET Standard class libraries: They are a merger of the platform-specific and portable library concepts into a single model that provides the best of both.

1.7 SUMMARY

This chapter briefs about role of .NET framework while designing an application. Also, it gives you idea about components and design principles of .NET framework. It also states various languages used in .NET for designing any windows or web application. The CLR which is main component of .NET framework is discussed in this chapter and class library which provides various in built functions and support to execute the application smoothly is also discussed.

1.8 REFERENCES

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)

2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill

3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

1.9 QUESTIONS

1. Write a note on .NET framework.

2. What are the various components of .NET framework?

3. Write a note on CLR.

4. Explain the use of .NET class library.

5. Explain the terms CTS, CLS, and JIT.

❄❄❄❄❄❄❄

2

C# LANGUAGE BASICS

Unit Structure :

2.0 INTRODUCTION

C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework. C# is used to develop web apps, desktop apps, mobile apps, games, and much more. C# is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, are said to be in the same class.

2.1 COMMENTS

Comments are used in a program to help us in understanding a piece of code. Comments are completely ignored by the compiler.

In C#, there are 2 types of comments:

- Single Line Comments (//)
- Multi Line Comments (/* */)

Single Line Comments

Single line comments start with a double slash //. The compiler ignores everything after // to the end of the line. For example,

```
int a = 7 + 9; // Adding 7 and 9
```

Multi Line Comments

Multi line comments start with /* and ends with */. Multi line comments can span over multiple lines.

```
/*    This is a first Program in C#.
      This basic program prints Hello World.
*/
using System;
namespace HelloWorld
{
      class Program
      {
            public static void Main(string[] args)
            {
                  // Prints Hello World
                  Console.WriteLine("Hello World!");
            }
      }
}
```

2.2 VARIABLE

A variable is the name of a memory location. It is used to store data. Its value can be changed and it can be reused many times. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory, and the set of operations that can be applied to the variable.

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet and underscore only. It can't start with a digit.
- No white space is allowed within the variable name.
- A variable name must not be any reserved word or keyword e.g. char, float etc.

Defining Variables

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

You can initialize a variable at the time of definition as −

```
int i = 100;
```

Valid variable names:

```
int s;
int _s;
int s05;
```

Invalid variable names:

```
int 5;
int x y;
int double;
```

Example

```
using System;
namespace VariableDefinition {
  class Program {
    static void Main(string[] args) {
      short a;
      int b ;
      double c;
      a = 10;
      b = 20;
      c = a + b;
      Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
      Console.ReadLine();
    }
  }
}
```

2.3 DATA TYPES

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

The variables in C#, are categorized into the following types −

- Value types
- Reference types
- Pointer types

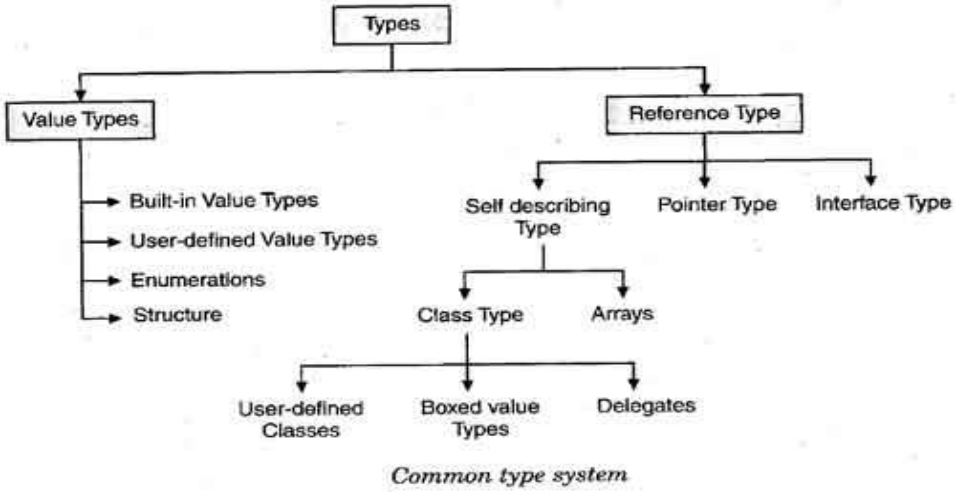


Fig. 2.1 classification of data types

Value Type - Value type variables can be assigned a value directly. The value types directly contain data. Some examples are int, char, and float, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an int type, the system allocates memory to store the value.

Type	Represents
bool	Boolean value
byte	unsigned integer
ushort	16-bit unsigned integer
uint	32-bit unsigned integer
ulong	64-bit unsigned integer
char	unicode character
decimal	Decimal values
double	Double precision floating point
float	Single precision floating point
sbyte	8-bit Signed integer
short	16-bit signed integer
int	32-bit Signed integer
long	64-bit Signed integer

Reference Type

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables. In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.

Example of built-in reference types are: object, dynamic, and string.

Object Type

The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class. The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.

When a value type is converted to an object type, it is called boxing and on the other hand, when an object type is converted to a value type, it is called unboxing.

Dynamic Type

We can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time. Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

String Type

The String Type allows us to assign any string values to a variable. The string type is an alias for the System.String class. It is derived from object type. The value for a string type can be assigned using string literals in two forms: quoted and @quoted.

Pointer Type

Pointer type variables store the memory address of another type.

2.4 VARIABLE OPERATIONS

Operators are symbols that are used to perform operations on operands. Operands may be variables and/or constants.

For example, in 2+3, + is an operator that is used to carry out addition operation, while 2 and 3 are operands.

Operators are used to manipulate variables and values in a program. C# supports a number of operators that are classified based on the type of operations they perform.

There are following types of operators to perform different types of operations in C# language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Unary Operators

- Ternary Operators
- Misc Operators

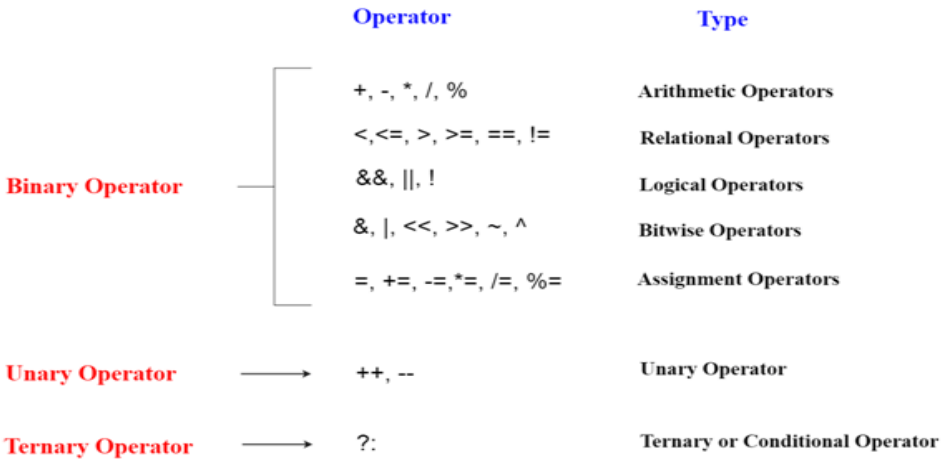


fig 2.2 types of operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

C# Arithmetic Operators		
Operator	Operator Name	Example
+	Addition Operator	6 + 3 evaluates to 9
-	Subtraction Operator	10 - 6 evaluates to 4
*	Multiplication Operator	4 * 2 evaluates to 8
/	Division Operator	10 / 5 evaluates to 2
%	Modulo Operator (Remainder)	16 % 3 evaluates to 1

Relational Operators

Relational operators are used to check the relationship between two operands. If the relationship is true the result will be true, otherwise, it will result in false.

C# Relational Operators		
Operator	Operator Name	Example
==	Equal to	6 == 4 evaluates to false
>	Greater than	3 > -1 evaluates to true
<	Less than	5 < 3 evaluates to false
>=	Greater than or equal to	4 >= 4 evaluates to true
<=	Less than or equal to	5 <= 3 evaluates to false
!=	Not equal to	10 != 2 evaluates to true

Logical Operators

Logical operators are used to perform logical operations such as and, or, not. Logical operators operate on boolean expressions (true and false) and return boolean values.

C# Logical operators				
Operand 1	Operand 2	OR (||)	AND (&&)	NOT(!) operand 1
true	true	true	true	false
true	false	true	false	false
false	true	true	false	true
false	false	false	false	true

Bitwise Operators

Bitwise and bit shift operators are used to perform bit level operations on integer (int, long, etc) and boolean data. These operators are not commonly used in real life situations.

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Operator	Description	Example
&	Binary AND Copies a bit if it exists in both operands	A = 14, B = 11 (A & B) = 10
|	Binary OOOR Copies a bit if it exists in wither operand	A = 14, B = 11 (A | B) = 15
^	Binary XOR Copies a bit if it is set in one operand but not in both	A = 14, B = 11 (A ^ B) = 5
~	Binary ones complement Flipping bits	A = 26 (~A) = 229
<<	Binary left shift	A = 42

	Left operands value is moved left by number of bits specified by right operand	A << 1 = 84 A << 2 = 168
>>	Binary right shift Left operands value is moved right by number of bits specified by right operand	A = 42 A >> 1 = 21 A << 2 = 10

Assignment Operators

Operator	Description	Example
=	Assignment Assigns value from right to left side	C = A + B
+=	Add AND assignment Adds right operand to left operand and assign result to left operand	C += A C = C + A
-=	Subtract AND assignment Subtracts right operand from left operand and assign result to left operand	C -= A C = C - A
*=	Multiply AND assignment Multiplies right operand with left operand and assign result to left operand	C *= A C = C * A
/=	Divide AND assignment Divides left operand with right operand and assign result to left operand	C /= A C = C / A

Unary operator

The unary operators operates on a single operand.

C# unary operators		
Operator	**Operator Name**	**Description**
+	Unary Plus	Leaves the sign of operand as it is
-	Unary Minus	Inverts the sign of operand
++	Increment	Increment value by 1
--	Decrement	Decrement value by 1
!	Logical Negation (Not)	Inverts the value of a boolean

Ternary Operator

The ternary operator ? : operates on three operands. It is a shorthand for if-then-else statement. Ternary operator can be used as follows:

variable = Condition? Expression1 : Expression2;

If the expression stated by Condition is true, the result of Expression1 is assigned to variable.

If it is false, the result of Expression2 is assigned to variable.

int number = 10;

string result;

result = (number % 2 == 0)? "Even Number" : "Odd Number";

Console.WriteLine("{0} is {1}", number, result);

Miscellaneous Operators

sizeof()	Returns size of a data type	sizeof(int)
typeof()	Returns type of a class	typeof(StreamReader)
&	Returns address of an variable	&a
*	Pointer to a variable	*a
?:	Conditional expression	if condition is true ? Then x : else y
is	Determines whether an object is of a certain type	if(Ford is Car) checks if ford is an object of the car class
as	Cast without raising an exception if the cast fails	object obj = new StringReader(“Hello”) StringReader r = obj as StringReader

2.5 OBJECT BASED MANIPULATION

- A C# program consists of the following parts:
  - Namespace declaration
  - A class
  - Class methods
  - Class attributes
  - A Main method
  - Statements and Expressions
  - Comments

```
using System;
namespace HelloWorldApplication
{
   class HelloWorld
   {
      static void Main(string[] args)
      {
         /* my first program in C# */
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

- The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.
- The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.
- The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.
- The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main** method states what the class does when executed.
- The next line /*...*/ is ignored by the compiler and it is put to add **comments** in the program.
- The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**
- *WriteLine* is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- The last line **Console.ReadKey();** is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

Conditional logic

Ans. Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

C# provides following types of decision making statements.

if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.

- **if Statement**

  An if statement consists of a boolean expression followed by one or more statements.

  Syntax

  if(boolean_expression)

  {

  /* statement(s) will execute if the boolean expression is true */

  }

  If the boolean expression evaluates to true, then the block of code inside the if statement is executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement(after the closing curly brace) is executed.

Flow Diagram

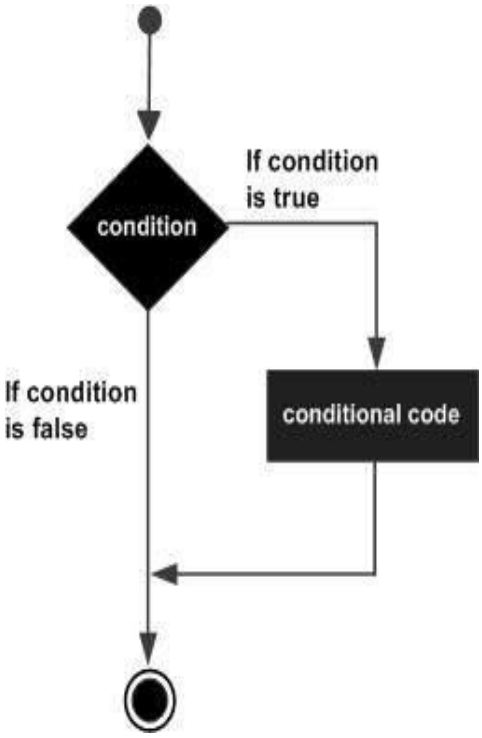


fig. 2.3 if - flow chart

```
using System;
namespace DecisionMaking
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 10;
         /* check the boolean condition using if statement */
         if (a < 20)
         {
            /* if condition is true then print the following */
            Console.WriteLine("a is less than 20");
         }
         Console.WriteLine("value of a is : {0}", a);
         Console.ReadLine();
      }
   }
}
```

o/p :- a is less than 20; value of a is : 10

- **if...else Statement**

  An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

  Syntax

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
   /* statement(s) will execute if the boolean expression is false */
}
```

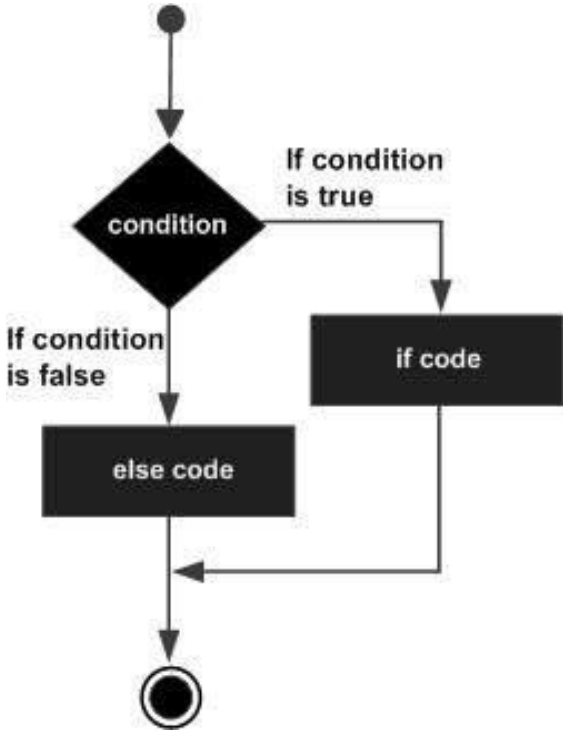


fig 2.4 if…else – flow chart

```
using System;
namespace DecisionMaking
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 100;

         /* check the boolean condition */
         if (a < 20)
         {
            /* if condition is true then print the following */
            Console.WriteLine("a is less than 20");
         }
         else
         {
            /* if condition is false then print the following *,
            Console.WriteLine("a is not less than 20");
         }
         Console.WriteLine("value of a is : {0}", a);
         Console.ReadLine();
      }
   }
}
```

O/p

a is not less than 20;

value of a is : 100

- **Nested if Statements**

you can use one if or else if statement inside another if or else if statement(s).

Syntax

```
if( boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */
   if(boolean_expression 2)
   {
      /* Executes when the boolean expression 2 is true */
   }
}
```

Example

```
class Program
{
   static void Main(string[] args)
   {
      //* local variable definition */
      int a = 100;
      int b = 200;

      /* check the boolean condition */
      if (a == 100)
      {
         /* if condition is true then check the following */
         if (b == 200)
         {
            /* if condition is true then print the following */
            Console.WriteLine("Value of a is 100 and b is 200");
         }
      }
      Console.WriteLine("Exact value of a is : {0}", a);
      Console.WriteLine("Exact value of b is : {0}", b);
      Console.ReadLine();
   }
}
}
```

- O/p :-

  Value of a is 100 and b is 200

  Exact value of a is : 100

  Exact value of b is : 200

- **Switch Statement**

  A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Syntax

```
switch(expression) {
   case constant-expression  :
      statement(s);
      break; /* optional */
   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
   statement(s);
}
```

Flow Diagram

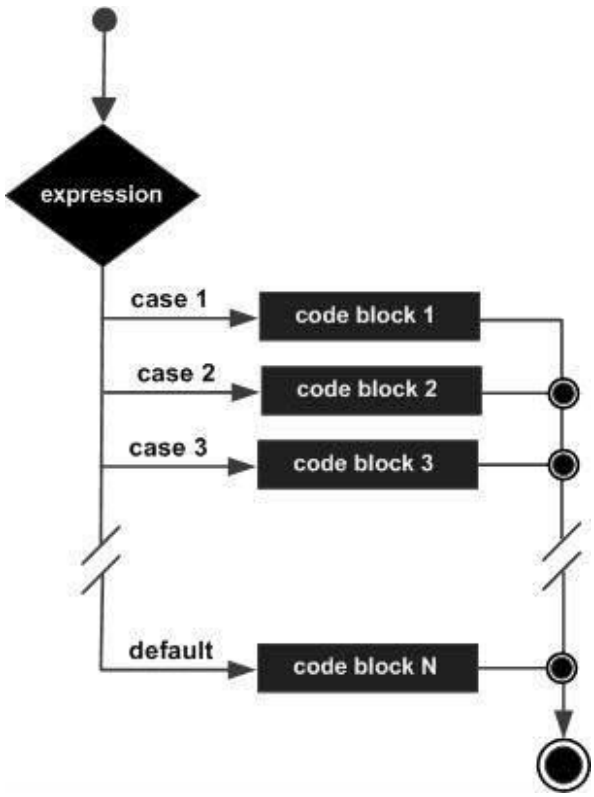


fig. 2.5 switch

```
using System;
namespace DecisionMaking
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         char grade = 'B';

         switch (grade)
         {
            case 'A':
               Console.WriteLine("Excellent!");
               break;
            case 'B':
            case 'C':
               Console.WriteLine("Well done");
               break;
            case 'D':
               Console.WriteLine("You passed");
               break;
            case 'F':
               Console.WriteLine("Better try again");
               break;
               default:
            Console.WriteLine("Invalid grade");
               break;
         }
         Console.WriteLine("Your grade is  {0}", grade);
         Console.ReadLine();
      }
   }
}
```

O/P :-

Well done

Your grade is B

Loops

A loop statement allows us to execute a statement or a group of statements multiple times.

Types of loop statements

- **while loop** :- It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
- **for loop** :- It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- **do...while loop** :- It is similar to a while statement, except that it tests the condition at the end of the loop body

While Loop

A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.

Syntax

```
while(condition)
{
   statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram

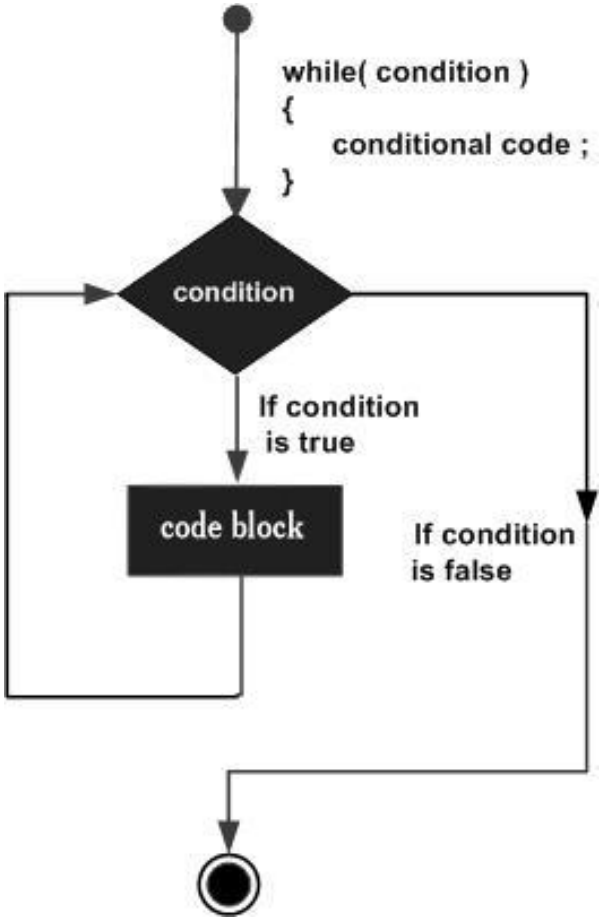


fig 2.6 while condition

```
using System;
namespace Loops
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 10;

         /* while loop execution */
         while (a < 20)
         {
            Console.WriteLine("value of a: {0}", a);
            a++;
         }
         Console.ReadLine();
      }
   }
}
```

O/P

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

```
for ( init; condition; increment )
{
   statement(s);
}
```

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again testing for a condition). After the condition becomes false, the for loop terminates.

Flow diagram

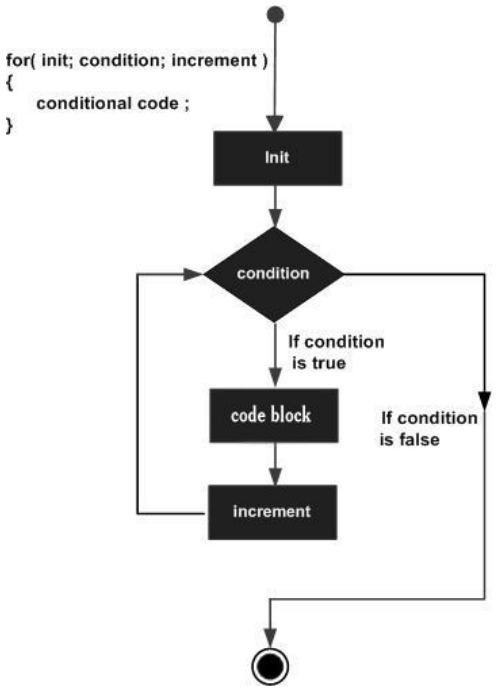


fig 2.7 for loop

```
using System;
namespace Loops
{
   class Program
   {
      static void Main(string[] args)
      {
         /* for loop execution */
         for (int a = 10; a < 20; a = a + 1)
         {
            Console.WriteLine("value of a: {0}", a);
         }
         Console.ReadLine();
      }
   }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Do…while loop

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

```
do
{
   statement(s);

}while( condition );
```

The conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

Flow Diagram

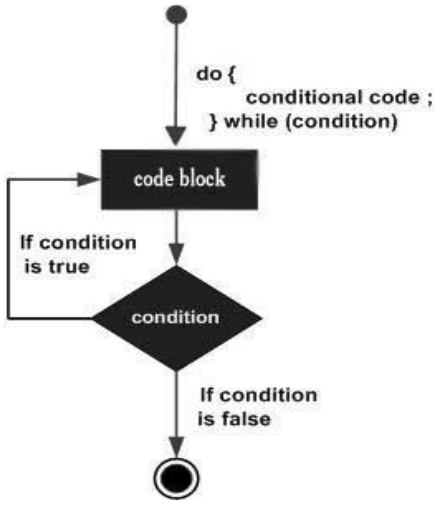


fig 2.8 do…while loop flow chart

```
using System;
namespace Loops
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 10;

         /* do loop execution */
         do
         {
            Console.WriteLine("value of a: {0}", a);
            a = a + 1;
         }
         while (a < 20);
         Console.ReadLine();
      }
   }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C# provides the following control statements.

break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Break Statement

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- If we are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Flow Diagram

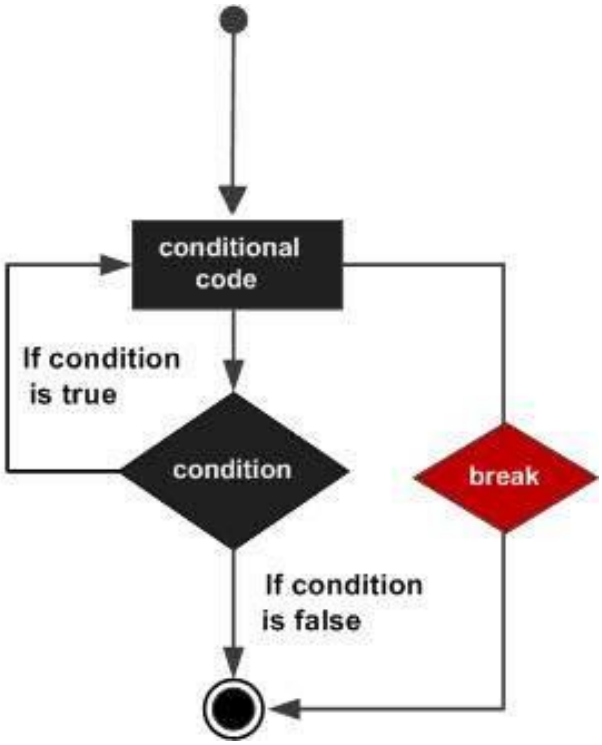


fig 2.8 break statement flow chart

Example

```
using System;
namespace Loops
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 10;

         /* while loop execution */
         while (a < 20)
         {
            Console.WriteLine("value of a: {0}", a);
            a++;
            if (a > 15)
            {
               /* terminate the loop using break statement */
               break;
            }
         }
         Console.ReadLine();
      }
   }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Continue Statement

It forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue statement causes the conditional test and increment portions of the loop to execute.

The while and do...while loops, continue statement causes the program control passes to the conditional tests.

Flow Diagram

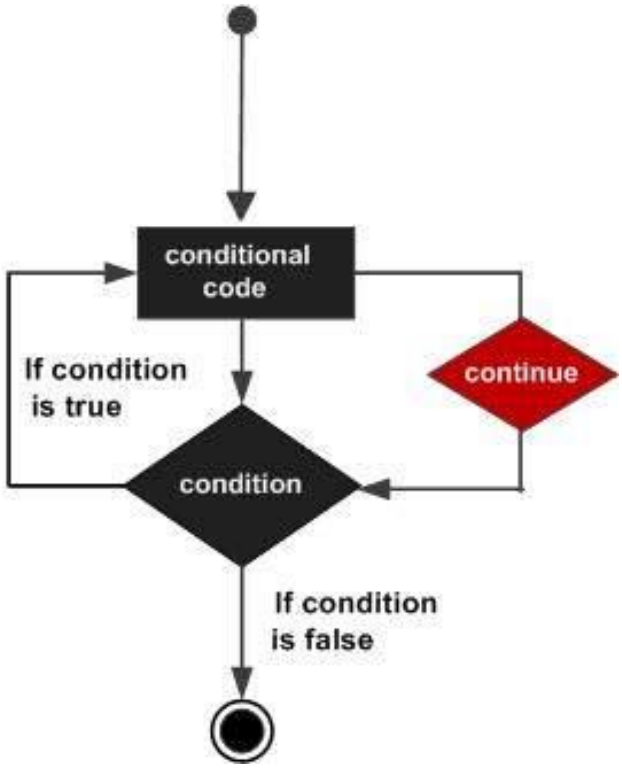


fig 2.9 continue statement flow chart

Example

```
using System;
namespace Loops
{
   class Program
   {
      static void Main(string[] args)
      {
         /* local variable definition */
         int a = 10;

         /* do loop execution */
         do
         {
            if (a == 15)
            {
               /* skip the iteration */
               a = a + 1;
               continue;
            }
            Console.WriteLine("value of a: {0}", a);
            a++;
         }
         while (a < 20);
         Console.ReadLine();
      }
   }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Methods

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

Defining Methods in C#

Syntax

<Access Specifier><Return Type><Method Name>(Parameter List) {

Method Body

}

- Access Specifier − Determines the visibility of a variable or a method from another class.
- Return type − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- Method name − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- Parameter list − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- Method body − It contains the set of instructions needed to complete the required activity.

Calling Methods in C#

```
namespace CalculatorApplication {

   class NumberManipulator {

      public intFindMax(int num1, int num2) {

int result;

       if (num1 > num2)

          result = num1;

        else

          result = num2;

        return result;

      }
```

```
    static void Main(string[] args) {

int a = 100;

int b = 200;

int ret;

NumberManipulator n = new NumberManipulator();

    ret = n.FindMax(a, b);

Console.WriteLine("Max value is : {0}", ret );

Console.ReadLine();

    }

  }

}
```

When the above code is compiled and executed, it produces the following result −

```
Max value is : 200
```

Classes

Class is a blueprint for a data type. It does not actually define any data, but it defines what the class name means. That is, what an object of the class consists of and what operations can be performed on that object. Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

Defining a Class

A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces.

```
<access specifier> class  class_name {

  // member variables

<access specifier><data type> variable1;

<access specifier><data type> variable2;

  // member methods
```

```
<access specifier><return type> method1(parameter_list) {
    // method body
  }
}
```

- Access specifiers specify the access rules for the members as well as the class itself.
- Data type specifies the type of variable, and return type specifies the data type of the data the method returns, if any.

```
namespace BoxApplication {
  class Box {
    public double length;   // Length of a box
    public double breadth;  // Breadth of a box
    public double height;   // Height of a box
  }
  class Boxtester {
    static void Main(string[] args) {
      Box Box1 = new Box();   // Declare Box1 of type Box
      double volume = 0.0;    // Store the volume of a box here
      Box1.height = 5.0;
      Box1.length = 6.0;
      Box1.breadth = 7.0;
      volume = Box1.height * Box1.length * Box1.breadth;
Console.WriteLine("Volume of Box1 : {0}",  volume);
Console.ReadKey();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Volume of Box1 : 210
```

2.6 CALL BY VALUE AND CALL BY REFERENCE

Two ways of passing parameters to methods are call by value and call by reference

Call by Value

Call by value method copies the value of an argument into the formal parameter of that function. Therefore, changes made to the parameter of the main function do not affect the argument. In this parameter passing method, values of actual parameters are copied to the function's formal parameters, and the parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

```
using System;
class Test
{
static void Change(int a)
{
  a = 5;
  Console.WriteLine(a);
}
static void Main(string[] args)
{
  int n = 10;
  Change(n);
  Console.WriteLine(n);
}
}
```

Call by Reference

Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument. In this method, the memory allocation is the same as the actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameter, and the modified value will be stored at the same address.

```
using System;
class Test
{
static void Change(ref int a)
```

```
{
  a = 5;
  Console.WriteLine(a);
}
static void Main(string[] args)
{
  int n = 10;
  Change(ref n);
  Console.WriteLine(n);
}
}
```

2.7 C# CONSTRUCTORS

A special member function of a class that is executed whenever we create new objects of that class. A constructor has exactly the same name as that of class and it does not have any return type.

```
namespace LineApplication {

   class Line {
      private double length;   // Length of a line

      public Line() {
         Console.WriteLine("Object is being created");
      }

      public void setLength( double len ) {
         length = len;
      }

      public double getLength() {
         return length;
      }

      static void Main(string[] args) {
         Line line = new Line();

         // set line length
         line.setLength(6.0);
         Console.WriteLine("Length of line : {0}", line.getLength());
         Console.ReadKey();
      }
   }
}
```

Polymorphism(Function Overloading)

Multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. We cannot overload function declarations that differ only by return type.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Method_overloading32
{
    class shape
   {
        public void Area(int side)
         {
            int squarearea = side * side;
            Console.WriteLine("The Area of Square is :" + squarearea);
         }
        public void Area(int length, int breadth)
         {

            int rectarea = length * breadth;
            Console.WriteLine("The Area of Rectangle is :" + rectarea);
         }

        public void Area(double radius)
        {
            double circlearea = 3.14 * radius * radius;
            Console.WriteLine("The Area of Circle is :" + circlearea);
        }

    }
```

```
    class Program
    {
        static void Main(string[] args)
        {

            shape s = new shape();
            s.Area(10);
            s.Area(10, 20);
            s.Area(10.8);
            Console.ReadKey();

        }
    }
}
```

Namespace

A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace

```
namespace namespace_name {
   // code declarations
}
```

Example:-

```
namespace first_space {
   class namespace_cl {
      public void func() {
Console.WriteLine("Inside first_space");
      }
   }
}
namespace second_space {
   class namespace_cl {
      public void func() {
Console.WriteLine("Inside second_space");
      }
   }
}
class TestClass {
   static void Main(string[] args) {
first_space.namespace_cl fc = new first_space.namespace_cl();
second_space.namespace_clsc = new second_space.namespace_cl();
fc.func();
sc.func();
Console.ReadKey();
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Inside first_space
Inside second_space
```

Assemblies

An Assembly is a basic building block of .Net Framework applications. It is basically a compiled code that can be executed by the CLR. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or exe depending upon the project that we choose.

An assembly is a file that is automatically generated by the compiler upon a successful compilation of every .NET application. It is generated only once for an application and upon each subsequent compilation, the assembly gets updated. An Assembly contains Intermediate Language (IL) code, which is similar to Java byte code. In the .NET language, it consists of metadata. Metadata enumerates the features of every "type" inside the assembly or the binary. In addition to metadata, assemblies also have a special file called Manifest. It contains information about the current version of the assembly and other related information.

Assemblies are basically the following two types:

- Private Assembly
- Shared Assembly

Private Assembly

It is an assembly that is being used by a single application only. Suppose we have a project in which we refer to a DLL so when we build that project that DLL will be copied to the bin folder of our project. That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

Shared Assembly

Assemblies that can be used in more than one project are known to be shared assembly. Shared assemblies are generally installed in the GAC. Assemblies that are installed in the GAC are made available to all the .Net applications on that machine.

Inheritance

Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and speeds up implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **baseclass**, and the new class is referred to as the **derived class**.

A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.

Consider a base class Shape and its derived class Rectangle −

```
namespace InheritanceApplication {
   class Shape {
      public void setWidth(int w) {
         width = w;
      }
      public void setHeight(int h) {
         height = h;
      }
      protected int width;
      protected int height;
   }
   // Derived class
   class Rectangle: Shape {
      public intgetArea() {
         return (width * height);
      }
   }
   class RectangleTester {
      static void Main(string[] args) {
         Rectangle Rect = new Rectangle();
Rect.setWidth(5);
Rect.setHeight(7);
Console.WriteLine("Total area: {0}",  Rect.getArea());
Console.ReadKey();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Total area: 35
```

Static members

When we declare a member of a class as static, it means no matter how many objects of the class are created, there is only one copy of the static member.

The keyword static implies that only one instance of the member exists for a class. Static variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it. Static variables can be initialized outside the member function or class definition. You can also initialize static variables inside the class definition.

```
namespace StaticVarApplication {
  class StaticVar {
    public static intnum;
    public void count() {
num++;
    }
    public intgetNum() {
      return num;
    }
  }
  class StaticTester {
    static void Main(string[] args) {
StaticVar s1 = new StaticVar();
StaticVar s2 = new StaticVar();
      s1.count();
       s1.count();
       s1.count();
      s2.count();
       s2.count();
       s2.count();
Console.WriteLine("Variable num for s1: {0}", s1.getNum());
Console.WriteLine("Variable num for s2: {0}", s2.getNum());
Console.ReadKey();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Variable num for s1: 6
Variable num for s2: 6
```

2.8 CASTING OBJECTS

Type conversion is converting one type of data to another type. It is also known as Type Casting.

In C#, type casting has two forms −

- **Implicit type conversion** –It supports conversions from derived classes to base classes. Implicit conversion : It is done automatically by compiler; no data will be lost; includes conversion of a smaller data type to a larger data types; safe type conversion. Ex:-

  int smallnum = 654667;

  long bigNum = smallnum;

- **Explicit type conversion** − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator. Conversion of larger data type to smaller data type; information might be lost or conversion might not be succeed for some reasons. This is an un-safe type conversion.

  long bigNum = 654667;

  int smallnum = (int)bigNum;

Partial Classes

A partial class splits the definition of a class over two or more source files. We can create a class definition in multiple files but it will be compiled as one class.

Suppose we have a "Person" class. That definition is divided into the two source files "Person1.cs" and "Person2.cs". Then these two files have a class that is a partial class. We compile the source code then create a single class.

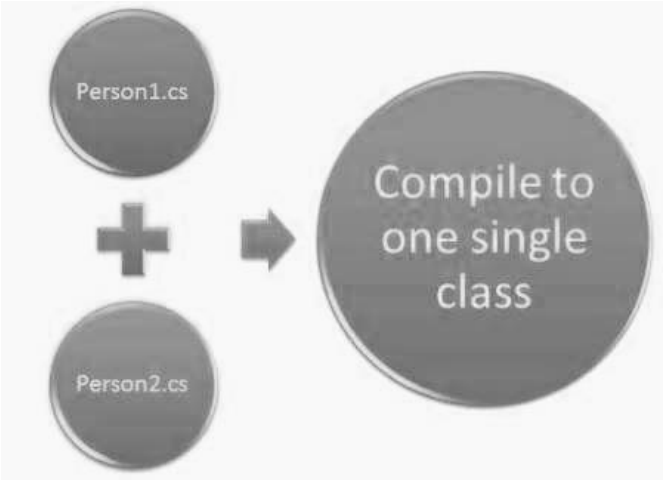


fig. 2.10 partial class

Advantages of a partial class

We can separate User Interface design code and business logic code so that it is easy to read and understand.

1. When working with automatically generated source, the code can be added to the class without having to recreate the source file.

2. More than one developer can simultaneously write the code for the class.

3. We can maintain our application better by compacting large classes. Suppose we have a class that has multiple interfaces so we can create multiple source files depending on interface implements. It is easy to understand and maintain an interface implemented on which the source file has a partial class.

```
publicinterface IRegister
{
   //Register realted function
}
publicinterface ILogin
{
   //Login related function
}

//UserRegister.cs file
publicpartial classUser : IRegister, ILogin
{
   //implements IRegister interface
}

//UserLogin.cs file
publicpartial classUser
{
   //implements ILogin interface
}
```

2.9 SUMMARY

This chapter briefs about basics of C# while developing a web application. This chapter discusses variable declaration, various operators used in C#, conditional logics and looping concepts in C#. It also focuses on implementation of OOPs concept with respect to C#. This chapter tell you about data types supported by asp.net web applications.

2.10 REFERENCES

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)

2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill

3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

2.11 QUESTIONS

1. What are the data types supported by C#?
2. Describe various decision making statements in C#.
3. Explain in brief Loop Control Statements
4. How to define and call a method in C#?
5. Write a note on C# classes.
6. What do you mean by namespace? Explain it with an example.
7. What are the advantages of inheritance? Explain it with suitable example.
8. What is the use of static members? Explain it with an example.
9. How can be object casting or data casting is possible in C#?
10. What is partial class? State its advantages.

❄❄❄❄❄❄❄

3

ASP.NET

Unit Structure :

3.0 INTRODUCTION

ASP.NET is an open source web framework, created by Microsoft, for building modern web apps and services with .NET. ASP.NET is cross platform and runs on Windows, Linux, macOS, and Docker. ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications and web services. It provides fantastic integration of HTML, CSS and JavaScript. It was first released in January 2002. It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.

3.1 CREATING WEBSITES

1- Start -> All Programs -> Visual Studio 2008

2- Now go to File Menu -> New -> Web Site

3- Under Visual Studio Installed Template-> Choose ASP.NET WEB SITE -> Choose File System from the location combo box -> Set the path by the browse button - > Choose the language from the Language ComboBox (Visual C# , Visual Basic , J #)

Choose Visual C#

4 - Click on the OK Button

5- Tab named Design in the bottom of this page.

Click on this tab and you will see a blank web page where you can drag any control from the toolbox (which is in the left side of this window).

6- If you are not able to see the Toolbox window just go to View -> Choose Toolbox.

7 - Drag a button on the blank page and now click on the Source tab.

```
<%@ Page Language="C#"
AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>

<asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

8- Now you can drag any control from the toolbox to the blank web page and you can set the properties of these controls from the property window

9- If You are not able to see the property window just go to View - > Select Properties Window. A new window will open on the right side of this page…

10-Select the control for which you want to set the properties

Go to properties window and set the properties like Button Name

11- If you want to change the Button name which you have added in your web site then select the button and right click on the button select properties and the select the — Text Property from the property window and give any name which you want it will change the button name.

3.2 ANATOMY OF A WEB FORM

ASP.NET applications are generally divided into multiple Web pages. Every Web page in an ASP.NET application shares a common set of resources and configuration settings.

Each ASP.NET application is executed inside a separate application domain. These application domains ensure that even if a Web application causes a fatal error, it does not affect other applications that are currently running on the same computer.

Each Web application is a separate entity that has its own set of data. It can be described as a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory on a Web server.

The following listing shows an example of a relatively simple Web Forms page.

HelloSimple.aspx

<%-- Example of the @ Page directive --%>
<%@ Page Language="c#" ClassName="Hello" %>

<html>

<head>

<script runat="server">

public void SayHello(Label1.Text = "Hello, " + name + "!")

public void Page_Load(object sender, System.EventArgs e)

if (IsPostBack)

if (NameTextBox.Text != "" Name = NameTextBox.Text
SayHello() </script> </head>

3.3 UNDERSTANDING PAGE ELEMENTS

HelloSimple.aspx shows examples of many of the elements that you can use in an ASP.NET Web Form, including server-side comments, the @ Page directive, static HTML, a server-side <script> code-declaration block containing both event handlers and methods, and several ASP.NET server controls.

Element	Description
Static HTML tags	These standard HTML elements are treated by ASP.NET as literal controls, and are rendered to the client browser as represented in the source file.
HTML comments	Syntax: <!-- -->. HTML comments allow descriptive text to be added to a page. This text is sent to the client but is not rendered by the browser.
Directives	Directives, such as the @ Page directive, provide the ASP.NET runtime with information about how to process the page. Using directives, you can control such ASP.NET

Element	Description
	features as session state, wiring up of events, and output caching, as well as importing namespaces and registering custom controls for use within a page.
Server-side code	Code can be contained in either server-side <script> code declaration blocks or <% %> render blocks. ASP.NET supports server-side code in any language that targets the runtime.
Event handlers	Event handlers are procedures in <script> code declaration blocks that handle page or server control events, such as Page_Load or control Click events. Most ASP.NET code should be written in or called from event handlers, rather than being written in render blocks.
<script> code declaration blocks	These blocks are used to contain page-level procedures and to declare variables that are global to the page. Executable code, other than global variable declarations in code declaration blocks, must be contained within a procedure declaration. Server-side code declaration blocks must have the runat="server" attribute, as shown in HelloSimple.aspx.
<% %> render blocks	These blocks are used to contain executable code not contained within procedures. Overuse of render blocks can result in code that is difficult to read and maintain.
Client-side <script> blocks	These blocks are used to contain script code to be executed on the client, usually in response to a client-side event. Choice of language (set by the language attribute) is dictated by the languages supported by the target browser. JavaScript is the most common choice for cross- browser compatibility in client scripts.
Server-side comments	Syntax: <%-- --%>. Server-side comments allow descriptive text to be added to a page. Unlike HTML comments, this text is not sent to the client.
User controls	These are custom controls that are defined declaratively in files with the .ascx extension. They provide a simple and straightforward mechanism for reuse of UI and UI-related code, and can contain most of the same elements as Web Forms pages.
ASP.NET server controls	This set of built-in controls provides ASP.NET developers with a programming model that mimics that of Microsoft Visual Basic. Controls are added to a page, and programmers write code to handle events raised by users' interaction with the controls at runtime. ASP.NET provides two sets of built-in controls: the HTML controls,

Element	Description
	which provide a 1-to-1 mapping of server-side controls for most HTML elements; and the Web controls, which provide a set of controls that are very similar to the Visual Basic UI controls. Note that some server controls, such as the TextBox and Button controls, must be placed within a server-side <form>, or an exception will be raised.
Custom server controls	Custom server controls are another mechanism for reuse in ASP.NET. They're defined in class files (.cs or .vb files) and are precompiled into managed assemblies before use.

3.4 THE PAGE DIRECTIVE

ASP.NET directives are instructions to specify optional settings, such as registering a custom control and page language. These settings describe how the web forms (.aspx) or user controls (.ascx) pages are processed by the .Net framework.

The syntax for declaring a directive is:

<<%@ directive_name attribute=value [attribute=value] %>

The Page directive defines the attributes specific to the page file for the page parser and the compiler. Page Directives are commands. These commands are used by the compiler when the page is compiled. The page directive gives ASP.NET basic information about how to compile the page. It indicates the language you're using for your code and the way you connect your event handlers. If you're using the code-behind approach, the page directive also indicates where the code file is located and the name of your custom page class.

The basic syntax of Page directive is:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" Trace="true" %>

The attributes of the Page directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page_Load.
Buffer	The Boolean value that enables or disables HTTP response buffering.
ClassName	The class name for the page.
ClientTarget	The browser for which the server controls should render content.

CodeFile	The name of the code behind file.
Debug	The Boolean value that enables or disables compilation with debug symbols.
Description	The text description of the page, ignored by the parser.
EnableSessionState	It enables, disables, or makes session state read-only.
EnableViewState	The Boolean value that enables or disables view state across page requests.
ErrorPage	URL for redirection if an unhandled page exception occurs.
Inherits	The name of the code behind or other class.
Language	The programming language for code.
Src	The file name of the code behind class.
Trace	It enables or disables tracing.
TraceMode	It indicates how trace messages are displayed, and sorted by time or category.
Transaction	It indicates if transactions are supported.
ValidateRequest	The Boolean value that indicates whether all input data is validated against a hardcoded list of values.

The Doctype :

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "https://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

In an ordinary, non-ASP.NET web page, the doctype occupies the very first line. In an ASP.NET web form, the doctype gets second place, and appears just underneath the page directive. The doctype indicates the type of markup (for example, HTML or XHTML) that you're using to create your web page. Technically, the doctype is optional, but Visual Studio adds it automatically. This is important, because depending on the type of markup you're using there may be certain tricks that aren't allowed. For example, strict XHTML doesn't let you use HTML formatting features that are considered obsolete and have been replaced by CSS.

code-behind class

Code Behind refers to code for ASP.NET page which is contained within a separate class file. It is composed in a different class record that can have the extension of .aspx.cs or .aspx.vb relying upon the language used. It allows a clean separation of HTML from the presentation logic. In the code-behind file, you create a class (which can be any class derived from the Page class) that serves as the base class for the web page you create in the .aspx

file. This relationship between your class and the web page is established by a Page directive at the top of the .aspx file:

< %@ Page inherits="NewPage" % >

The inherits attribute identifies the class created in the code-behind file from which this .aspx file will derive. One major point of Code Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.

If you use code-behind class files with .aspx pages, you can separate the presentation code from the core application logic (or code-behind). The code-behind class file is compiled so that it can be created and used as an object. This allows access to its properties, its methods, and its event handlers.

Right-click on the .aspx page, and then click **View Code**. The code-behind file opens in the editor. In the code-behind file, add the following code to the Page_Load event handler:

```
private void Page_Load(object sender, System.EventArgs e)
{
   Label1.Text = "(Precompiled): Page_Load fired!";
}
```

3.5 ADDING EVENT HANDLERS

An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events. For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it. All GUI applications are incomplete without enabling actions. Events can also be generated without user interactions. Event handlers are methods in an object that are executed in response to some events occurring in the application. Events in ASP.NET raised at the client machine, and handled at the server machine. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.

The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

Event Arguments

ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.

Syntax :-

```
private void EventName (object sender, EventArgs e);
```

To create an event handler for the default event

In Design view, double-click the page or double-click the control for which you want to create a default event handler.

To create an event handler in the Properties window

In Design view, select the control for which you want to create an event handler. In **Properties**, click the events symbol. The **Properties** window displays a list of events for the selected control.

In the box next to an event name, do one of the following:

Double-click to create a new event handler for that event. The designer will name the handler using the convention controlID_event.

Type the name of the handler to create.

In the drop-down list, select the name of an existing handler.

The drop-down list displays a list of methods that have the correct signature for the event.

3.6 ANATOMY OF AN ASP.NET APPLICATION

ASP.NET File Types

.asax	Typically a Global.asax file that represents the application class and contains event handlers that run at various points in the application life cycle.
.ascx	A Web user control file that defines a custom functionality that you can add to any ASP.NET Web Forms page.
.ashx	A handler file that is invoked in response to a Web request in order to generate dynamic content.
.asmx	An XML Web services file that contains classes and methods that can be invoked by other Web applications.
.aspx	An ASP.NET Web Forms page that can contain Web controls and presentation and business logic.
.cd	A class diagram file.
.config	A configuration file contains XML elements that represent settings for ASP.NET features.
.cs, .vb	Source code files (.cs or .vb files) that define code that can be shared between pages
.csproj, .vbproj	A project file for a Visual Studio Web-application project.
.dll	A compiled class library file (assembly).
.master	A master page that defines the layout for other Web pages in the application.

.mdb	An Access database file.
.resources, .resx	A resource file that contains resource strings that refer to images, localizable text, or other data.
.sitemap	A sitemap file that defines the logical structure of the Web application. ASP.NET includes a default sitemap provider that uses sitemap files to display a navigational control in a Web page.
.sln	A solution file for a Visual Studio project.

ASP.NET Web Folders

The asp.net application folder contains list of specified folder that you can use of specific type of files or content in an each folder. The root folder structure is as following

- BIN
- App_Code
- App_GlobalResources
- App_LocalResources
- App_WebReferences
- App_Data
- App_Browsers
- App_Themes

Bin Directory

It is contains all the precompiled .Net assemblies like DLLs that the purpose of application uses. The *Bin* folder is used for keeping assemblies inside it. We can access those as a reference from anywhere of our web application. Use of *Bin* folder comes into the picture if we use any class library within our web application.

App_Code Directory

It contains source code files like .cs or .vb that are dynamically compiled for use in your application. These source code files are usually separate components or a data access library. As its name suggests, the *App_Code* Folder stores classes, typed data sets, etc. All the items that are stored in *App_Code* are automatically accessible throughout the application. If we store any class files (like *.cs* or *.vb*) it compiles them automatically. It automatically creates type data sets from *.xsd* (XML schema) files, and creates XML web service proxy classes from WSDL.

App_GlobalResources Directory

It contains to stores global resources that are accessible to every page. The *App_GlobalResource* folder can be read from any page or code that is anywhere in the web site. Global resources must be stored in

the *App_GlobalResource* folder at the root of the application. We should use the *App_GlobalResource* folder when we need a single resource for multiple web pages. We can define ASP.NET control properties by manually associating them with resources in global resource files. You can add a global resource file by right clicking on the *App_GlobalResource* folder and clicking on **Add Items**. Add *.resx* files as resources.

App_LocalResources Directory

It serves the same purpose as app_globalresources, except these resources are accessible for their dedicated page only. Local resources are specific to a single web page, and should be used for providing multilingual functionality on a web page. Local resources must be stored in the *App_LocalResource* subfolder of the folder containing the web page. Because you might have local resources for every page in your web application, you might have *App_LocalResource* subfolders in every folder.

App_WebReferences Directory

It stores reference to web services that the web application uses. As the name suggests, the *App_WebReference* folder contain references to any web services. If we have added any web services with our web application, they go automatically into the *App_WebReference* folder, in the same way as in windows applications, if we added any DLLs, they would go under the *Reference* folder.

App_Data Directory

It is reserved for data storage and also mdf files, xml file and so on. The *App_Data* folder is used as a data storage for the web application. It can store files such as *.mdf*, *.mdb*, and XML. It manages whole application's data centrally. It is accessible from anywhere in your web application.

App_Browsers Directory

It contains browser definitions stored in xml files. These xml files define the capabilities of client side browsers for different rendering actions. The *App_Browser* folder contains browser information files (*.browser* files). These files are XML based files which are used to identify the browser and browser capabilities.

App_Themes Directory

It contains collection of files like .skin and .css files that are used to improve application’s look and feel appearance. If you want to give your web sites a consistent look, then you need to design themes for your web application. The *App_Themes* folder contains all such themes. An *App_Theme* folder can contain two subfolders; one for CSS files and the other for skin files. When we add an *App_Theme* folder, a subfolder with name *Theme1* will be automatically created. We can change the name of the theme folder as per our requirements.

3.7 ADVANTAGES OF ASP.NET APPLICATION FOLDERS

- We can maintain resources (classes, images, code, databases, themes) in an organized manner, which allows us to develop and maintain sites easily
- All files and folders are accessible through the application
- We can add as many files as required
- Files are compiled dynamically when required

3.8 SUMMARY

This chapter briefs about anatomy of a web form. This chapter discusses about how code behind technique helps web developers developing the web pages and creating and using events of asp.net. It also focuses on various ASP.NET folders and file types.

3.9 REFERENCES

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)

2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill

3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

3.10 QUESTIONS

1. Explain the anatomy of a web form.
2. Explain various types of asp.net file types.
3. Explain various types of asp.et web folders.

❄❄❄❄❄❄❄

4

HTML SERVER CONTROLS

Unit Structure :

4.0 INTRODUCTION

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing. The HTML controls such as the header tags, anchor tags, and input elements are not processed by the server but are sent to the browser for display. By default, HTML elements on an ASP.NET Web page are not available to the server. These components are treated as simple text and pass through to the browser. They are specifically converted to a server control by adding the attribute runat="server" and adding an id attribute to make them available for server-side processing.

For example, consider the HTML input control:

<input type="text" size="40">

It could be converted to a server control, by adding the runat and id attribute:

<input type="text" id="testtext" size="40" runat="server">

All the HTML Server controls can be accessed through the **Request** object. The HTML server controls have the same HTML output and the same properties as their corresponding HTML tags. In addition, HTML server controls provide automatic state management and server-side events. HTML server controls offer the following advantages:

- The HTML server controls map one to one with their corresponding HTML tags.
- When the ASP.NET application is compiled, the HTML server controls with the runat=server attribute are compiled into the assembly.

- Most controls include an OnServerEvent for the most commonly used event for the control. For example, the <input type=button> control has an OnServerClick event.
- The HTML tags that are not implemented as specific HTML server controls can still be used on the server side; however, they are added to the assembly as HtmlGenericControl.
- When the ASP.NET page is reposted, the HTML server controls keep their values.

The following are HTML server controls that are available in ASP.NET:-

- HtmlAnchor Control
- HtmlButton Control
- HtmlForm Control
- HtmlImage Control
- HtmlInputButton Control
- HtmlInputCheckBox Control
- HtmlInputFile Control
- HtmlInputHidden Control
- HtmlInputImage Control
- HtmlInputRadioButton Control
- HtmlInputText Control
- HtmlSelect Control
- HtmlTable Control
- HtmlTableCell Control
- HtmlTableCell Control
- HtmlTextArea Control

Example

Here, we are implementing an HTML server control in the form.

// htmlcontrolsexample.aspx

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="html controlsexample.aspx.cs" Inherits="asp.netexample.htmlcontrolsexample " %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

```
</head>
<body>
<form id="form1" runat="server">
<div>
<input id="Text1" type="text" runat="server"/>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Butto
n1_Click"/>
</div>
</form>
</body>
</html>
```

This application contains a code behind file.

// htmlcontrolsexample.aspx.cs

```
using System;
namespace asp.netexample
{
public partial class htmlcontrolsexample : System.Web.UI.Page
   {
protected void Button1_Click(object sender, EventArgs e)
      {
string a = Request.Form["Text1"];
         Response.Write(a);
      }
   }
}
```

4.1 VIEW STATE

View state is used automatically by the ASP.NET page framework to persist information that must be preserved between postbacks. This information includes any non-default values of controls. You can also use view state to store application data that is specific to a page. View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back. It is maintained internally as a hidden field in the form of an encrypted value and a key. Default enables the View State for a page. When the browser renders the HTML markup, the current state and the page values are retained and serialized into base64-encoded strings. The View State methods differ from the cache and cookies

because the cookies are accessible from all the pages on your website, while the View State values are non-transferable, and thus you cannot access from different pages.

Features of View State

- Retain the control value on a page without storing them in a user profile or session state.
- Store page values and control properties that you set or define on a page.
- Create a custom View State Provider to store the view page information in a SQL Server Database or another database.

View State can handle several data objects. A few of them are:

- String
- Boolean Value
- Array Object
- Array List Object
- Hash Table
- Custom type Converters

It's possible to store other data types too. The only condition is that you must compile the class with the Serializable attribute to serialize the values for View State.

Advantages of View State

Simplicity and Ease of Use - There is no need for complex codes and logical thinking to implement ViewState. It is simple and makes storing form data between page submissions easy.

Flexibility - It is easy to enable, configure, and disable View State properties on a control-by-control basis. Hence the developer may also choose to implement it at a page level or a control level.

Server-Independent - There are absolutely no server resources required to use View State. It is contained in a structure within the page load.

Enhanced Security - The View State values go through hashing, encoding, and compression for Unicode implementation.

How to Enable and Disable View State?

View State can be enabled and disabled for a single control as well as at the page level. Set the EnableViewState attribute of a single control to false to disable View State for that control.

TextBox1.EnableViewState=false;

To disable the View State for an entire page, set the page directive's EnableViewState to false as seen below:

<%Page Language="C#" EnableViewState="false";

You must use the same property and set it to "True" to enable the same.

Example

Store the value in viewstate : ViewState[“name”]= “SK College”;

Retrieve information from viewstate : string value=ViewState[“name”].ToString();

Open visual studio and design web form with two button control, a textbox and a label control.

Here, we have two button control one for a clear textbox value and second one for a retrieve the same textbox value after clearing it. Before clearing textbox value store it in ViewState[“name”] and after clearing it get value from ViewState[“name”] and display in label while clicking display value button.

```
protected void btnclear_Click(object sender, EventArgs e)
{
   ViewState["name"] = txtname.Text;
   txtname.Text = "";
}
protected void btndisplay_Click(object sender, EventArgs e)
{
   lbl.Text = ViewState["name"].ToString();
}
```

4.2 THE HTMLCONTROL CLASS

The HtmlControl class is the basis for all HTML server controls in Visual Basic. You don't use it directly—instead, you use classes derived from it. The classes derived from the HtmlControl class are:

- HtmlContainerControl
- HtmlImage
- HtmlInputControl

The real purpose of the HtmlControl class is to provide a set of properties shared by all HTML server controls:

- Attributes— Holds all attribute name and value pairs for the server control's HTML element.
- Disabled— Gets/sets whether the disabled attribute is included when an HTML control is displayed in the browser.
- Style— Gets all cascading style sheet (CSS) properties for the specified HTML server control.

- TagName— Gets the element name that contains the runat=server attribute.

HtmlControl

The HtmlControl object is very important to HTML server controls. Because every property and method it has inherited is by every HTML server control. If you learn the properties and methods of the HtmlControl class, you have learned about 80% of the properties and methods of all the objects in the System.Web.UI.HtmlControls namespace. The HTML server controls have their own properties and methods, as well, but they all have the properties and methods contained in the HtmlControl object.

Property	Description
Attribute	Returns the object's attributes collection.
Disabled	A Boolean (true or false) value that you can get or set that indicates whether a control is disabled.
EnableViewState	A Boolean (true or false) value that you can get or set that indicates whether a control should maintain its viewstate.
ID	A string that you can get or set that defines the Identifier for the control.
Style	Returns the CSSStyleCollection for a control.
TagName	Returns the tag name of an element such as input or div.
Visible	A Boolean (true or false) value that you can get or set that indicates whether a control is rendered to HTML for delivery to the client's browser.

The Html Control Base Classes

The base class for all HTML controls is System.Web.UI.HtmlControls.HtmlControl. This exposes methods, properties, and events that are common to all HTML controls.

Member	Description
Attributes property	Returns a collection of all the attribute name/value pairs within the .aspx file for this control. Can be used to read and set nonstandard attributes (custom attributes that are not actually part of HTML) or to access attributes where the control does not provide a specific property for that purpose.
ClientID property	Returns the control identifier that is generated by ASP.NET.

Member	Description
Controls property	Returns a ControlCollection object containing references to all the child controls for this control within the page hierarchy.
Disabled property	Sets or returns a Boolean value indicating if the control is disabled.
EnableViewState property	Sets or returns a Boolean value indicating if the control should maintain its viewstate and the viewstate of any child controls when the current page request ends. The default is True.
ID property	Sets or returns the identifier defined for the control.
Page property	Returns a reference to the Page object containing the control.
Parent property	Returns a reference to the parent of this control within the page hierarchy.
Style property	References a collection of all the CSS style properties (selectors) that apply to the control.
TagName property	Returns the name of the element, for example a or div.
Visible property	Sets or returns a Boolean value indicating if the control should be rendered in the page output. Default is True.
DataBind method	Causes data binding to occur for the control and all of its child controls.
FindControl method	Searches within the current container for a specified server control.
HasControls method	Returns a Boolean value indicating if the control contains any child controls.
DataBinding event	Occurs when the control is being bound to a data source.

The HtmlContainerControl Class

Any HTML control that requires a closing tag inherits from the HtmlContainer class. For example, elements such as <a>, <form>, and always use a closing tag. Elements such as <img> and <input> are used only as stand-alone tags. HtmlAnchor, HtmlForm, and HtmlGenericControl classes inherit from HtmlContainerControl.

The HtmlContainer control adds two properties to those defined in HtmlControl.

InnerHtml The HTML content between the opening and closing tags of the control.

InnerText The text content between the opening and closing tags of the control

The HtmlContainerControl class serves as the abstract base class that defines the methods, properties, and events available to all HTML server controls that can act as a container (in HTML terms, these elements all require a closing tag). This class is the base class for the HtmlTableCell, HtmlTable, HtmlTableRow, HtmlButton, HtmlForm, HtmlAnchor, HtmlGenericControl, HtmlSelect, and HtmlTextArea classes, all of which share these properties:

Constructors

Name	Description
HtmlContainerControl()	Create a new HtmlContainerControl
HtmlContainerControl(Element)	Create a new control.

The HtmlInputControl Class

The HtmlInputControl class serves as the abstract base class that defines the methods, properties, and events common to all HTML input controls, such as the <input type="text">, <input type="submit">, and other elements that the user can enter data into. The classes derived from the HtmlInputControl class are the HtmlInputText, HtmlInputButton, HtmlInputCheckBox, HtmlInputImage, HtmlInputHidden, HtmlInputFile, and HtmlInputRadioButton classes, all of which share the following properties:

Name— Gets/sets a unique name for the input control.

Value— Gets/sets the contents of an input control.

Type— Gets the type of an input control.

The following controls, which are based on the HTML INPUT element, are available on the HTML tab of the Toolbox:

- **Input (Button)** control: INPUT type="button" element
- **Input (Checkbox)** control: INPUT type="checkbox" element
- **Input (File)** control: INPUT type="file" element
- **Input (Hidden)** control: INPUT type="hidden" element
- **Input (Password)** control: INPUT type="password" element
- **Input (Radio)** control: INPUT type="radio" element
- **Input (Reset)** control: INPUT type="reset" element
- **Input (Submit)** control: INPUT type="submit" element
- **Input (Text)** control: INPUT type="text" element

Unlike other HTML elements, if you convert an HTML INPUT element to an ASP.NET server control, it is not created as an instance of the HtmlInputControl class. You cannot create an instance of the HtmlInputControl class directly. Instead, this class is inherited by the classes listed in the table below.

The following table lists the type that is used to instantiate INPUT elements as ASP.NET server controls if the markup contains the attribute runat="server" and an id attribute.

Server control	Type
Button control	HtmlInputButton
CheckBox control	HtmlInputCheckBox
File Field control	HtmlInputFile
Hidden control	HtmlInputHidden
Password control	HtmlInputPassword
Radio Button control	HtmlInputRadioButton
Reset Button control	HtmlInputReset
Submit Button control	HtmlInputSubmit
Text Field control	HtmlInputText

4.3 PAGE CLASS

- Page class represents an .aspx file, also known as a Web Forms page, requested from a server that hosts an ASP.NET Web application.Every web page is a custom class that inherits from the system.web.UI.Page control. By inheriting form this class, page class acquires a number of properties that our code can use.
- **IsPostBack** :- This Boolean property indicates whether this is the first time the page is being run (False) or whether the page is being resubmitted in response to a control event, typically with stored view state information (True).
- **EnableViewState** :- When set to False, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
- **Application** :- This collection holds information that's shared between all users in the website. For example, we can use the Application collection to count the number of times a page has been visited.
- **Session** :- holds information for a single user, so it can be used in different pages. For example, we can use the Session collection to store the items in the current user's shopping basket on an e-commerce website.

- **Cache** :- allows us to store objects that are time-consuming to create so they can be reused in other pages or for other clients. Improves performance of the web pages.
- **Request** :-HttpRequest object that contains information about the current web request.
- **Response** :-HttpResponse object that represents the response ASP.NET will send to the user's browser.
- **Server** :-HttpServerUtility object that allows us to perform a few miscellaneous tasks. For example, it allows us to encode text so that it's safe to place it in a URL or in the HTML markup of the page.
- **User** :- If the user has been authenticated, this property will be initialized with user information.

4.4 GLOBAL.ASAX

The Global.asax file, sometimes called as ASP.NET application file. It allows us to write code that responds to global application events. These events fire at various points during the lifetime of a web application, including when the application domain is first created. We can use this file to implement application security, as well as other tasks. The Global.asax file is placed in the root application directory. The.NET IDE automatically inserts it in all new ASP.NET projects. This file is an optional file. We can delete it, when we are not using it.

To add a Global.asax file to an application in Visual Studio, choose Website -> add new item and select the global application class file type. Then, click OK.

Ex:- The following Global.asax file reacts to the Application.EndRequest event, which happens just before the page is sent to the user :-

<%@ Application Language="VB" %>

<script runat="server">

Sub Application_EndRequest(ByVal sender As Object, ByVal e As EventArgs)

Response.write("<hr>This page was served at " &DateTime.Now.ToString())

End Sub

</script>

4.5 WEB.CONFIG

Behavior of an ASP.NET application is affected by different settings in the configuration files:

- machine.config
- web.config

machine.config file contains default and machine-specific value for all supported settings.

Machine settings are controlled by the system administrator and applications are generally not given access to this file.

Every web application includes a web.config file that configures fundamental settings—everything from the way error messages are shown to the security settings that lock out unwanted visitors. ASP.NET stores settings in a human-readable XML format using configuration files such as machines.config and web.config. An application however, can override the default values by creating web.config files in its roots folder. The web.config file is a subset of the machine.config file. If the application contains child directories, it can define a web.config file for each folder. Scope of each configuration file is determined in a hierarchical top-down manner. Any web.config file can locally extend, restrict or override any settings defined on the upper level. Visual Studio generates a default web.config file for each project. An application can run without a web.config file, however, we cannot debug an application without a web.config file. The XML based web.config file is used to specify the application wide settings for the entire application. The web.config file is present in the root of the application's directory, although we can also have multiple web.config files, one for each subdirectory. In a Web.config file, sections can appear in the settings area that have not been declared in the declaration area if they are declared in a .config file at a higher level in the configuration hierarchy.

4.6 SUMMARY

This chapter briefs about use of html server controls while developing a web application. This chapter discusses various aspects of html server controls such as html control classes, events, html container control class, html input control class. It also focuses on role of page class, global.asax file and web.config file helps while developing web applications.

4.7 REFERENCES

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)

2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill

3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

4.8 QUESTIONS

1. Write a note on view state.
2. Write a note on HtmlControl class.
3. What is a page class? Explain the use of its properties.
4. What is the use of global.asax file?
5. Write a note on web.config file?

❄❄❄❄❄❄❄

5

WEB CONTROLS

Unit Structure :

In this chapter, you'll explore the basic web controls and their class hierarchy. You'll also delve deeper into ASP.NET's event handling and learn the details of the web page life cycle.

5.0 OBJECTIVES

In this chapter,

- you'll explore the basic web controls and their class hierarchy.
- you'll also delve deeper into ASP.NET's event handling and learn the details of the web page life cycle.

5.1 INTRODUCTION

Web Controls are small building blocks of the GUI(Graphical User Interface), which include labels, text box, buttons, etc which provide rich functionality in your pages.

You need web controls because HTML control corresponds directly to a single HTML element. Web controls, on the other hand, have no such restriction—they can switch from one element to another depending on how you're using them, or they can render themselves by using a complex combination of multiple elements.

Advantages of using Web Controls:

- They provide a rich user interface
- They provide a consistent object model
- They tailor their output automatically
- They provide high-level features

5.2 WEB CONTROL CLASSES

- Web control classes are defined in the **System.Web.UI.WebControls** namespace. They follow a slightly more tangled object hierarchy than HTML server controls.
- Figure shows most, but not all, of the web controls that ASP.NET provides.

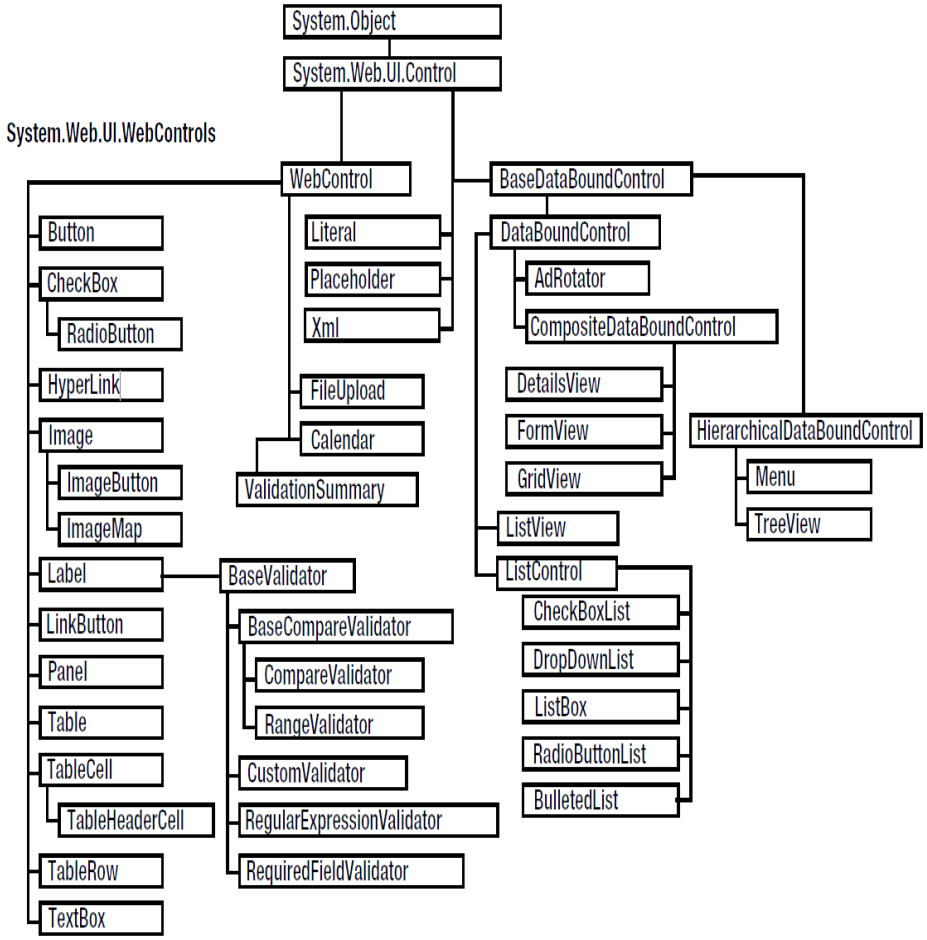


Figure 5.1: The web control hierarchy
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

5.3 WEBCONTROL BASE CLASS

- Web controls provides the properties, methods, and events that are common to all Web server controls.
- Most web controls begin by inheriting from the WebControl base class. This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with almost any web control, as described in Table

Properties of webcontrol classes

Property	Description
AccessKey	This property is used to set focus on web control.
BackColor	Sets the colors used for the background
ForeColor	Sets the colors used for the foreground. In most controls, the foreground color sets the text color.
BorderColor	Sets the colors used for the Border
BorderWidth	Specifies the size of the control border.
BorderStyle	One of the values from the BorderStyle enumeration, including Dashed, Dotted, Double, Groove, Ridge, Inset, Outset, Solid, and None.
Enabled	When set to false, the control will be visible, but it will not be able to receive user input or focus.
EnableViewState	This property state whether the view state of the control is maintained or not. Set this to false to disable the automatic state management for this control.
Font	Specifies the font used to render any text in the control
Height and Width	Specifies the width and height of the control.
ID	Specifies the name that you use to interact with the control in your code
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	This property is used to set Parent for web control.
TabIndex	A number that allows you to control the tab order. The control with a TabIndex of 0 has the focus when the page first loads.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the Web server control.
Visible	When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

Table 5.1: Properties of webcontrol classes
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Methods of web control classes:

Method	Description
AddAttributesToRender(HtmlTextWriter)	To add HTML attributes and styles in our web page that need to be rendered to the specified HtmlTextWriterTag
ClearChildState()	This method is used to deletes the view-state and control-state details of child controls of web control.
ClearChildViewState()	This method is used to delete the view-state information for all the child controls of web controls
CreateChildControls()	This method is used in creating child controls.
DataBind()	This method is used to binds a data source to the web control and all its child controls.
Dispose()	Enables a server control to perform final clean up before it is released from memory.
Focus()	Sets input focus to a control.
GetType()	Gets the Type of the current instance.
OnLoad(EventArgs)	Raises the Load event.
OnPreRender(EventArgs)	Raises the PreRender event.
OnUnload(EventArgs)	Raises the Unload event.
ToString()	Returns a string that represents the current object.

Table 5.2: Methods of web control classes
(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.webcontrol?view=netframework-4.8.1)

Events of Web Controls:

Name	Description
DataBinding	Occurs when the server control binds to a data source.
Disposed	This event occurs when a web control is released from memory
Init	Occurs when the server control is initialized
Load	Occurs when the server control is loaded into the Page object
PreRender	Occurs after the control object is loaded but prior to rendering.
Unload	Event Occurs when the server control is unloaded from memory.

Table 5.3: Events of Web Controls

(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.webcontrol?view=netframework-4.8.1)

5.4 LIST CONTROLS

- The list controls include the
  - ListBox,
  - DropDownList,
  - CheckBoxList,
  - RadioButtonList, and
  - BulletedList.
- They all work in essentially the same way but are rendered differently in the browser.
- The ListBox is a rectangular list that displays several entries, while the DropDownList shows only the selected item.
- The CheckBoxList and RadioButtonList are similar to the ListBox, but every item is rendered as a check box or option button, respectively.
- BulletedList is the only list control that isn't selectable. Instead, it renders itself as a sequence of numbered or bulleted items.
- All the selectable list controls provide a **SelectedIndex** property that indicates the selected row as a zero-based index.
- For example, if the first item in the list is selected, the SelectedIndex will be 0.

- Selectable list controls also provide an additional **SelectedItem** property, which allows your code to retrieve the **ListItem** object that represents the selected item.
- The ListItem object provides three important properties:
  - Text (the displayed content),
  - Value (the hidden value from the HTML markup), and
  - Selected (true or false depending on whether the item is selected).

5.4.1 ListBox

- This represents a list box control that allows single or multiple item selection.
- ListBox control has SelectionMode property that enables you to select multiple items from ListBox control. By default SelectionMode property is set as single.
- If you want to select multiple items from the ListBox, then set SelectionMode property value as Multiple and press Ctrl or Shift key when clicking more than one list item.

Example :

```
<asp:ListBox id="ListBox1" SelectionMode="Single" runat="server">
    <asp:ListItem>Item 1</asp:ListItem>
    <asp:ListItem>Item 2</asp:ListItem>
    <asp:ListItem>Item 3</asp:ListItem>
</asp:ListBox>
```

Common Properties of ListBox

Property	Description
Items	Gets the collection of items in the list control.
SelectionMode	This property will set the selection mode as single selection or multiple selection
Rows	This will determine the number of items shows in the list box.
SelectedIndex	Gets or sets the lowest ordinal index of the selected items in the list.
SelectedValue	Gets the value of the selected item in the list control, or selects the item in the list control that contains the specified value.

Table 5.4: Properties of ListBox
(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.listbox?view=netframework-4.8.1)

5.4.2 DropDownList

- This control permit user to select item from predefined list.
- It does not support for selecting multiple items at same time.

Example :

```
<asp:DropDownList ID="DropDownList1" runat="server" >
        <asp:ListItem Value="">Please Select</asp:ListItem>
        <asp:ListItem>item1 </asp:ListItem>
        <asp:ListItem> item2</asp:ListItem>
        <asp:ListItem> item3</asp:ListItem>
        <asp:ListItem> item4</asp:ListItem>
    </asp:DropDownList>
```

DropDownList carry same property like ListBox.

5.4.3 CheckBoxList

- CheckBoxList is generally used, when you want to select one or more options from given several choices.
- We can select more than one item from CheckBoxList control.
- The CheckBoxList control is easier for use, when you have set of options of checkboxes.

Example :

```
<asp:CheckBoxList id="checkboxlist1" runat="server">
      <asp:ListItem>Item 1</asp:ListItem>
      <asp:ListItem>Item 2</asp:ListItem>
      <asp:ListItem>Item 3</asp:ListItem>
 </asp:CheckBoxList>
```

Common properties:

Property	Description
Repeat Layout	Gets or sets a value that specifies whether the list will be rendered by using a table element, or list element.
RepeatColumns	Gets or sets the number of columns to display in the CheckBoxList control.
RepeatDirection	Gets or sets a value that indicates whether the control displays vertically or horizontally.

Table 5.5: Common properties of CheckBoxList
(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.checkboxlist?view=netframework-4.8.1)

5.4.4 RadioButtonList

- RadioButtonList Control is same as DropDownList but it displays a list of radio buttons that can be arranged either horizontally or vertically.
- You can select only one item from the given RadioButtonList of options.

Example :

```
<asp:RadioButtonList id="RadioButtonList1" runat="server">
        <asp:ListItem>Item 1</asp:ListItem>
        <asp:ListItem>Item 2</asp:ListItem>
        <asp:ListItem>Item 3</asp:ListItem>
 </asp:RadioButtonList>
```

Common properties:

Property	Description
Repeat Layout	Determines whether the radio buttons display in an HTML table.
RepeatColumns	It displays the number of columns of radio buttons.
RepeatDirection	The direction that the radio buttons repeat. By default RepeatDirection value is vertical. Possible values are Horizontal and Vertical.

Table 5.6: Common properties of RadioButtonList
(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.radiobuttonlist?view=netframework-4.8.1)

5.4.5 BulletedList

- BulletedList control is very rich in displaying the items in different styles. It dispalys the list either in unordered or ordered list.
- The default value of BulletStyle property is NotSet and rendered as in list of bulleted items.
- Possible values are as follows:

Circle, CustomImage, Disc, LowerAlpha, LowerRoman, NotSet, Numbered, Square, UpperAlpha, UpperRoman

- BulletedList control also supports the DisplayMode property that is used to modify the appearance of list items.
- Possible values are as follows: HyperLink, LinkButton, Text

Example:

<asp:BulletedList ID=”BulletedList1” runat=”server”>
</asp:BulletedList>

Common property:

Property	Description
BulletStyle	To set the style and looks of the bullet list this property is used.
FirstBulletNumber	In an ordered list, this sets the first value. Ex. If you set FirstBulletNumber to 3, the list might read 3,4,5 for Numbered.
DisplayMode	Determines whether the text of each item is rendered as text or a hyperlink.

Table 5.7: Common Property of BulletedList
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

5.5 TABLE CONTROLS

- Table class is used to build an HTML table.
- Table class is included in System.Web.UI.Controls namespace.
- Essentially, the Table control is built out of a hierarchy of objects. Each Table object contains one or more
- TableRow objects. In turn, each TableRow object contains one or more TableCell objects.

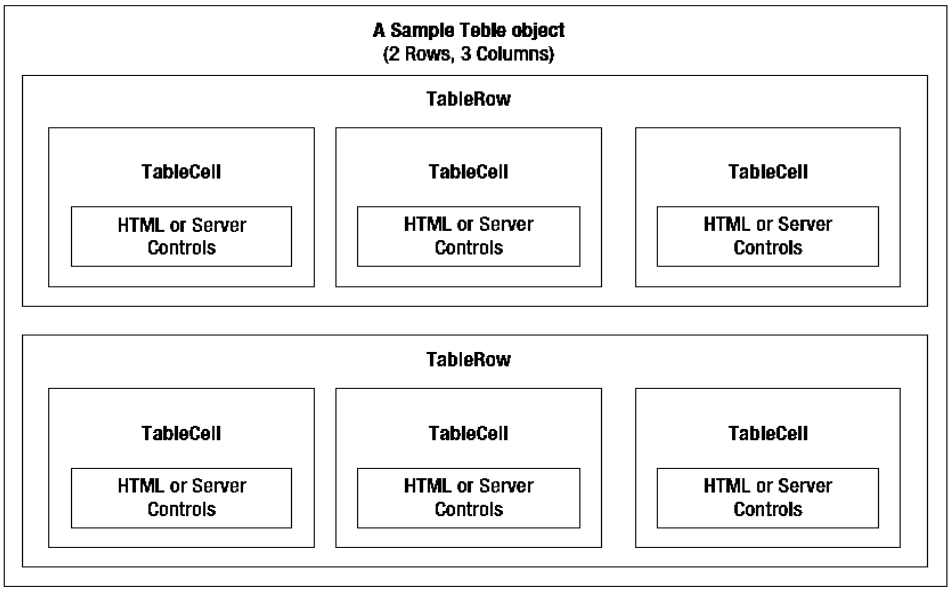


Figure 5.2: Table object
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Each TableCell object contains other ASP.NET controls or HTML content that displays information.
- We can create Table control in run-time as well as at design-time using the Visual studio.

Table control containment

Properties of Table class

Property	Description
Runat	This property state that the web control is a server control. For this purpose, we have to set value of runat to "server"
Rows	This property state group of rows in the table
Caption	This property is used to set title to table
CaptionAlign	This property is used to set alignment of the caption text.
CellPadding	This property is used to state the space between the cell walls and controls in table.
CellSpacing	This property is used to determines distance between cell of tables.
GridLines	This property is used to set gridline format in the table.
HorizontalAlign	Gets or sets the horizontal alignment of the Table control on the page.

Table 5.8: Table class Properties
(Ref: https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.table?view=netframework-4.8.1)

Example : Creating table at Design time

```
<asp:Table id="Table1" runat="server" CellPadding="10" GridLines="Both"
HorizontalAlign="Center">
    <asp:TableRow>
        <asp:TableCell>
            Row 0, Col 0
        </asp:TableCell>
        <asp:TableCell>
            Row 0, Col 1
        </asp:TableCell>
    </asp:TableRow>
```

```
    <asp:TableRow>
      <asp:TableCell>
        Row 1, Col 0
      </asp:TableCell>
      <asp:TableCell>
        Row 1, Col 1
      </asp:TableCell>
    </asp:TableRow>
  </asp:Table>
```

5.6 WEB CONTROL EVENTS AND AUTOPOSTBACK

- The previous chapter explained that one of the main limitations of HTML server controls is their limited set of useful events—they have exactly two.
- HTML controls that trigger a postback, such as buttons, raise a ServerClick event. Input controls provide a ServerChange event that doesn't actually fire until the page is posted back.
- Following Figure illustrates the order of events in page processing.

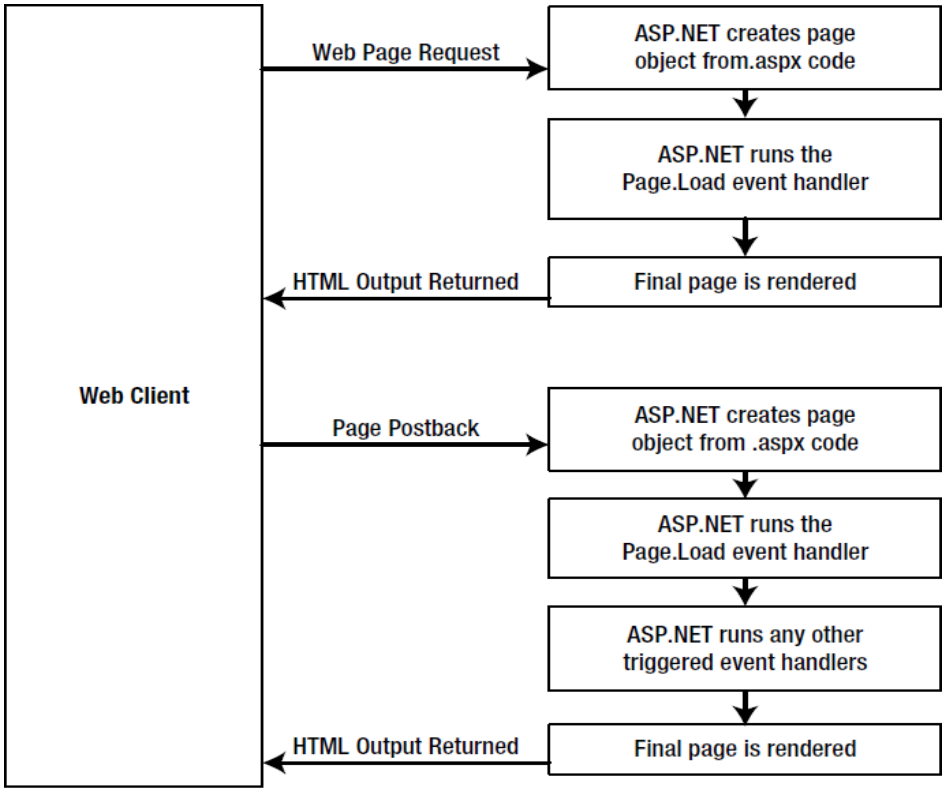


Figure 5.3: The page-processing sequence
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

The page-processing sequence

- When event occurs on client side some events such as Click event of a button take place immediately, because when clicked, the button post back the page.
- However, other actions do cause events but don't trigger a postback.
- For example, when user chooses a new item in a list or changes the text in text box.
- In these cases, without postback your code has no way to run.

ASP.NET handles this by giving you two options:

1. Wait until the next postback to react to the event.

   Like to react to SelectedIndexChanged event in a list, when user selects an item in a list, nothing happens immediately. But if user clicks a button to post back the page, two events fire: ButtonClick followed by ListBox.SelectedIndexChanged..

2. To use the automatic postback feature to force a control to post back the page immediately when it detects a specific user action. In this, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.

Event	**Web Controls That Provide It**	**Always Posts Back**
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

Table 5.9: Web Control Events
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- If you want to capture a change event (such as TextChanged, CheckedChanged, or SelectedIndexChanged) immediately, you need to set the control's AutoPostBack property to true.
- This way, the page will be submitted automatically when the user interacts with the control for example, picks a selection in the list, clicks a radio button or a check box, or changes the text in a text box and then moves to a new control).
- When the page is posted back, ASP.NET will examine the page, load all the current information, and then allow your code to perform some

extra processing before returning the page back to the user shown in following figure.

- Depending on the result you want, you could have a page that has some controls that post back automatically and others that don't.

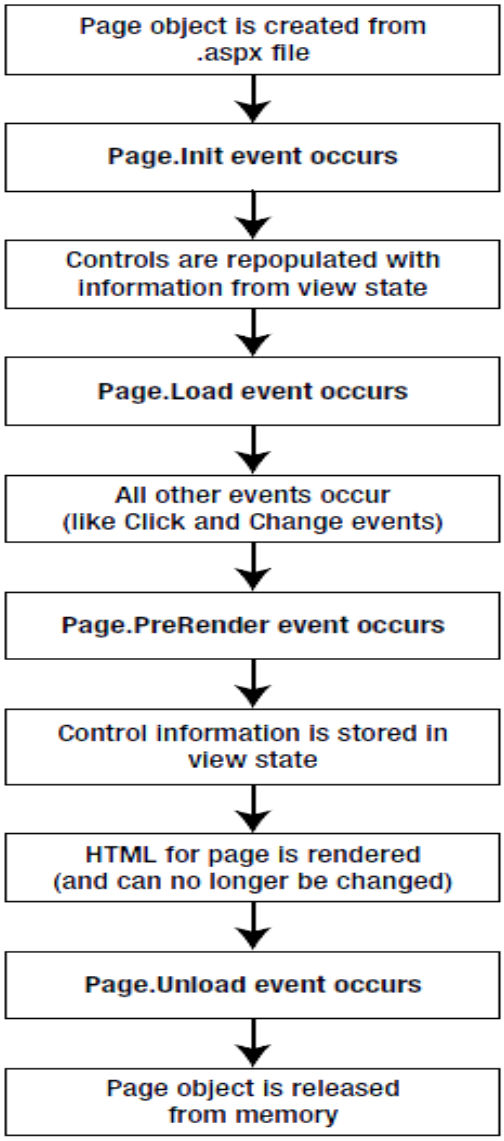


Figure 5.4: The postback processing sequence
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

5.7 PAGE LIFE CYCLE

To understand how web control events work, you need to have a solid understanding of the page life cycle.

Consider what happens when a user changes a control that has the AutoPostBack property set to true:

1. On the client side, the JavaScript __doPostBack function is invoked, and the page is resubmitted to the server.

2. ASP.NET re-creates the Page object by using the .aspx file.
3. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
4. The Page.Load event is fired.
5. The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
6. The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).
7. Finally, the Page.Unload event is fired.
8. The new page is sent to the client.

5.8 SUMMARY

This chapter introduced you to one of ASP.NET's richest features: web controls, and their object interface.

5.9 REFERENCE

- Beginning ASP.NET in C# by Matthew MacDonald
- https://learn.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols?view=netframework-4.8.1

5.10 QUESTIONS

1. Write a shot note on WebControls.
2. State the page life cycle with diagram.
3. Write and explain basic properties for table class.
4. Write and explain list control.
5. Write basic properties for webcontrol classes.

❄❄❄❄❄❄❄

6

STATE MANAGEMENT

Unit Structure :

In this chapter, you'll understand the use of State management in Web programming. Also, you will get to know about ViewState and Application State.

6.0 OBJECTIVES

- State management is a preserve state control and object in an application because web applications are stateless, which means a new web page object is re-created each time to serve request of client.
- This issue can be resoled by using State Management.

6.1 INTRODUCTION

- The most significant difference between programming for the Web and programming for the desktop is state management—how you store information over the lifetime of your application.
- This information can be as simple as a user's name or as complex as a stuffed-full shopping cart for an e-commerce store.
- Understanding these state limitations is the key to creating efficient web applications. In this chapter, you'll see how you can use ASP.NET's state management features to store information carefully and consistently.

6.2 VIEWSTATE

- ViewState is a important client side state management technique. ViewState is used to store user data on page at the time of post back of web page.
- ViewState does not hold the controls, it holds the values of controls.
- It does not restore the value to control after page post back.
- ViewState can hold the value on single web page, if we go to other page using response.redirect then ViewState will be null.
- When we require value of page variable to be maintained during page postback, we can use View state to store those value.
- "EnableViewState" property is used for both Page Level and Server contact level to manage the view state.
- Code for View state is like this,

  <input type="hidden" name="viewstate" id="viewstate1" value="123"/>
- This single hidden field contains all the view state values for all the page controls.

Example :

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

   <title>ViewState</title>

</head>

<body>

   <form id="form1" runat="server">

      <asp:TextBox runat="server" id="NameField" />

      <asp:Button runat="server" id="SubmitForm"
onclick="SubmitForm_Click"

            text="Submit & set name" />

      <asp:Button runat="server" id="RefreshPage" text="Just submit" />

      <br /><br />
```

```
    Name retrieved from ViewState: <asp:Label runat="server"
id="NameLabel" />
  </form>
</body>
</html>
```

And the CodeBehind:

```
using System;
using System.Data;
using System.Web;
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if(ViewState["NameOfUser"] != null)
      NameLabel.Text = ViewState["NameOfUser"].ToString();
    else
      NameLabel.Text = "Not set yet...";
  }
  protected void SubmitForm_Click(object sender, EventArgs e)
  {
    ViewState["NameOfUser"] = NameField.Text;
    NameLabel.Text = NameField.Text;
  }
}
```

- Try running the project, enter your name in the textbox and press the first button.
- The name will be saved in the ViewState and set to the Label as well.
- No magic here at all. Now press the second button.
- This one does nothing at all actually, it just posts back to the server.
- As you will notice, the NameLabel still contains the name, but so does the textbox.
- The first thing is because of us, while the textbox is maintained by ASP.NET it self. Try deleting the value and pressing the second button again.
- You will see that the textbox is now cleared, but our name label keeps the name, because the value we saved to the ViewState is still there!

Limitation of view state

1. Viewstate can be used only with single page.
2. It is storing the information of an hidden field, so it can be seen in source code in browser, hence it is not secure way.

6.3 CROSS-PAGE POSTING

- It's a technique that extends the postback mechanism, that one page can send the user to another page, complete with all the information for that page.
- "PostBackUrl" is the property name that provides the cross-page postback, which is defined by the IButtonControl interface and turns up in button controls such as ImageButton, LinkButton, and Button.
- To use cross-posting, we have to set PostBackUrl to the name of another web form.
- When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="CrossPage1.aspx.cs" Inherits="CrossPage1" %>
<html = "http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>CrossPage1</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            First Name:
            <asp:TextBox ID="txtFirstName"
runat="server"></asp:TextBox>
            <br/>
             Last Name:
             <asp:TextBox ID="txtLastName"
runat="server"></asp:TextBox>
             <br/>
             <br/>
           <asp:Button runat="server" Id="cmdPost"
PostBackUrl="CrossPage1.aspx" Text="Cross-Page Postback" /><br/>
          </div>
          </form>
</body>
</html>
```

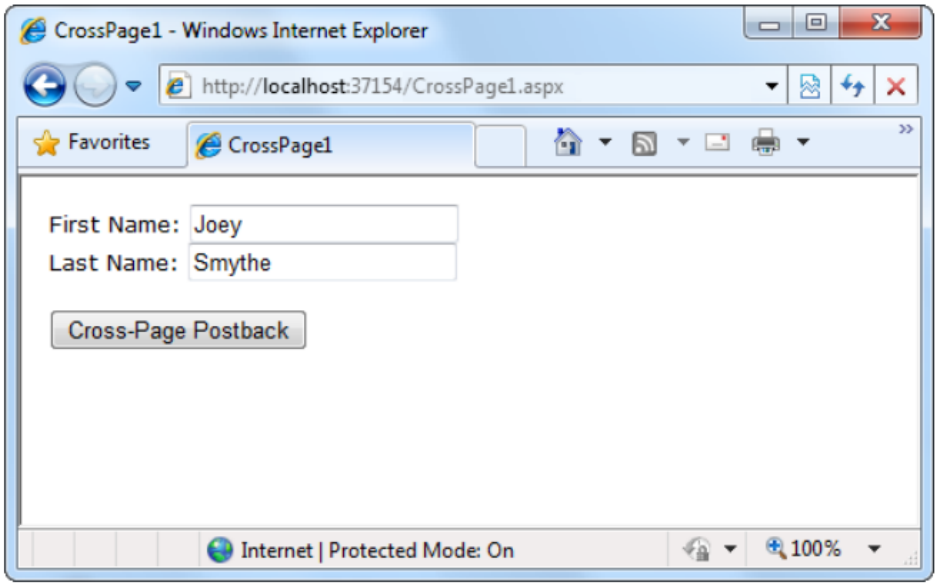


Figure 6.1: The starting point of a cross-page postback
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

6.4 QUERY STRING

- Another common approach is to pass information by using a query string in the URL.
- This approach is commonly found in search engines.
- For example, if you perform a search on the Google website, you'll be redirected to a new URL that incorporates your search parameters.
- Here's an example: http://www.google.ca/search?q=organic+gardening
- The query string is the portion of the URL after the question mark.
- In this case, it defines a single variable named q, which contains the string organic+gardening.
- There is a limitation of length of query string. So query string cannot be used to send very large data.
- Query string are visible to the user, so it should not be used to send sensitive information such as username, password without encryption.
- Request object of QueryString property is used to retrieve the query string.

Example:

- When the user chooses an item by clicking the appropriate item in the list, the user is forwarded to a new page.
- This page displays the received ID number.
- This provides a quick and simple query string test with two pages.

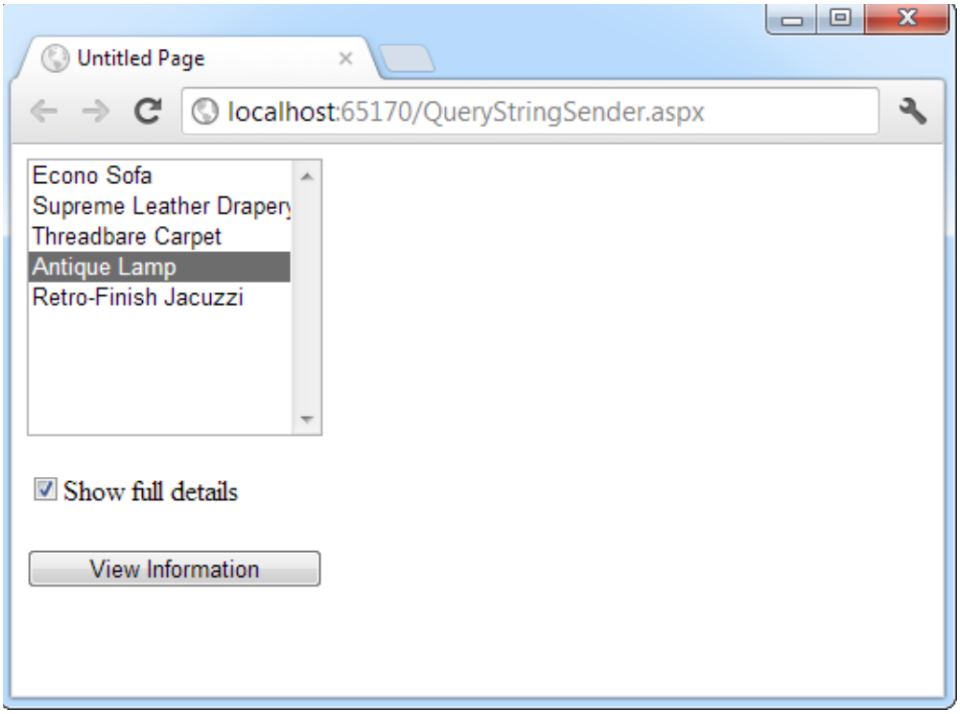


Figure 6.2: A query string sender
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Here's the code for the first page:

```
public partial class QueryStringSender : System.Web.UI.Page
 {
  protected void Page_Load(Object sender, EventArgs e)
    {
        if (!this.IsPostBack)
          {
            // Add sample values.
                lstItems.Items.Add("Econo Sofa");
                lstItems.Items.Add("Supreme Leather Drapery");
                lstItems.Items.Add("Threadbare Carpet");
                lstItems.Items.Add("Antique Lamp");
                lstItems.Items.Add("Retro-Finish Jacuzzi");
          }
       }
         protected void cmdGo_Click(Object sender, EventArgs e)
    {
        if (lstItems.SelectedIndex == -1)
        {
             lblError.Text = "You must select an item.";
         }
         else
```

```
        {
            // Forward the user to the information page,
            // with the query string data.
            string url = "QueryStringRecipient.aspx?";
            url += "Item=" + lstItems.SelectedItem.Text + "&";
            url += "Mode=" + chkDetails.Checked.ToString();
            Response.Redirect(url);
        }
    }
}
```

Here's the code for the recipient page

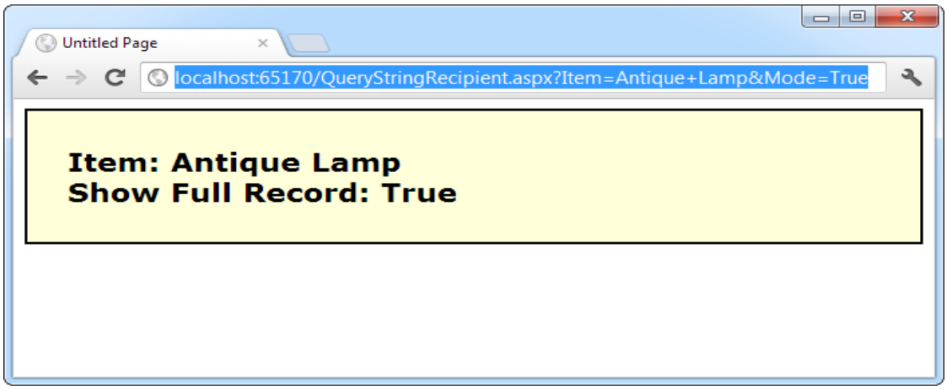


Figure 6.3: A query string recipient
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

```
public partial class QueryStringRecipient : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
            lblInfo.Text = "Item: " + Request.QueryString["Item"];
            lblInfo.Text+="Show Full Record: ";
            lblInfo.Text += Request.QueryString["Mode"];
     }
}
```

6.5 COOKIES

- Cookies provide another way to store information for later use.
- Cookies are small files that are created in the web browser's memory or on the client's hard drive.
- Cookies works transparently.

- To import cookies we should import the System.Net namespace so we can easily work with the appropriate types: using System.Net;
- Both the Request and Response objects provide a Cookies collection.

```
// Create the cookie object.
HttpCookie cookie = new HttpCookie("Preferences");
// Set a value in it.
cookie["LanguagePref"] = "English";
// Add another value.
cookie["Country"] = "US";
// Add it to the current web response.

Response.Cookies.Add(cookie);
```

- A cookie added in this way will persist until the user closes the browser and will be sent with every request. To create a longer-lived cookie, you can set an expiration date:
  ```
  // This cookie lives for one year.
  cookie.Expires = DateTime.Now.AddYears(1);
  ```
- You retrieve cookies by cookie name, using the Request.Cookies collection. Here's how you retrieve the preceding cookie, which is named Preferences:
  ```
  HttpCookie cookie = Request.Cookies["Preferences"];
  ```

Example:

The next example shows a typical use of cookies to store a customer name. To try this example, begin by running the page, entering a name, and clicking the Create Cookie button. Then close the browser, and request the page again. The second time, the page will find the cookie, read the name, and display a welcome message.

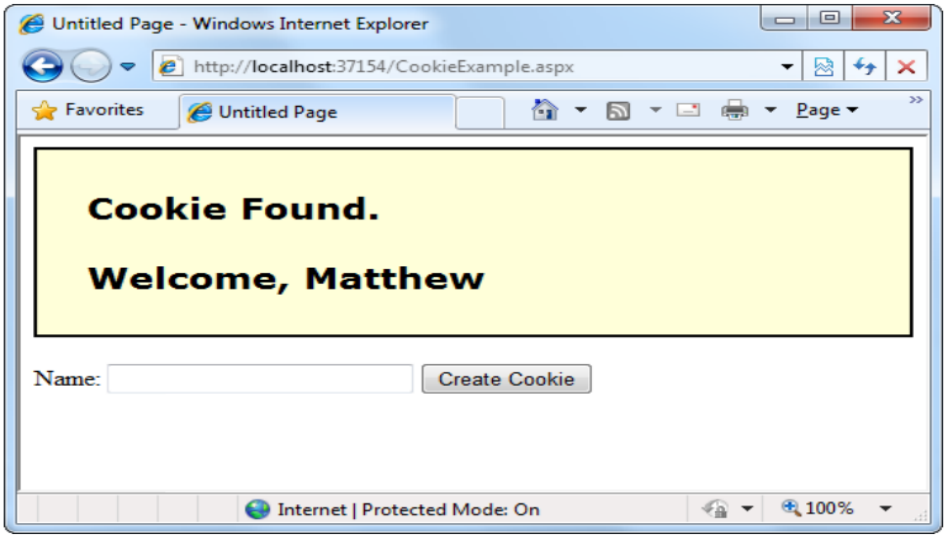


Figure 6.4: Displaying information from a custom cookie
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Here's the code for this page:

```
public partial class CookieExample : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
        HttpCookie cookie = Request.Cookies["Preferences"];
        if (cookie == null)
        {
            lblWelcome.Text = "<b>Unknown Customer</b>";
        }
        else
        {
            lblWelcome.Text = "<b>Cookie Found.</b>";
            lblWelcome.Text += "Welcome, " + cookie["Name"];
        }
}
protected void cmdStore_Click(Object sender, EventArgs e)
{
    // Check for a cookie, and create a new one only if
   // one doesn't already exist.
        HttpCookie cookie = Request.Cookies["Preferences"];
            if (cookie == null)
            {
                cookie = new HttpCookie("Preferences");
            }
            cookie["Name"] = txtName.Text;
            cookie.Expires = DateTime.Now.AddYears(1);
            Response.Cookies.Add(cookie);
            lblWelcome.Text = "<b>Cookie Created.<b/><br/>";
            lblWelcome.Text += "New Customer: " + cookie["Name"];
        }
}
```

6.6 SESSION STATE

- Session-state management is one of ASP.NET's premiere features. It allows you to store any type of data in memory on the server.
- The information is protected, because it is never transmitted to the client, and it's uniquely bound to a specific session.
- Every client that accesses the application has a different session and a distinct collection of information.

- Session state is ideal for storing information such as the items in the current user's shopping basket when the user browses from one page to another.

6.6.1 Session Tracking

- ASP.NET tracks each session by using a unique 120-bit identifier.
- ASP.NET uses a proprietary algorithm to generate this value, thereby guaranteeing that the number is unique.
- This ID is the only piece of session-related information that is transmitted between the web server and the client.
- When the client presents the session ID, ASP.NET looks up the corresponding session, retrieves the objects you stored previously, and places them into a special collection so they can be accessed in your code. This process takes place automatically.
- We can do this in two ways:
  1. Using cookies
  2. Using modified URLs

6.6.2 Using Session State

- You can interact with session state by using the System.Web.SessionState.HttpSessionState class, which is provided in an ASP.NET web page as the built-in Session object.
- For example, you might store a DataSet in session memory like this:
  Session["InfoDataSet"] = dsInfo;
- You can then retrieve it with an appropriate conversion operation:
  dsInfo = (DataSet)Session["InfoDataSet"];

6.6.3 HttpSessionState Members

Member	Description
Count	Provides the number of items in the current session collection
IsCookieless	Identifies whether the session is tracked with a cookie or modified URLs.
IsNewSession	Identifies whether the session was created only for the current request. If no information is in session state, ASP.NET won't bother to track the session or create a session cookie. Instead, the session will be re-created with every request.

Keys	Gets a collection of all the session keys that are currently being used to store items in the session-state collection.
Mode	Provides an enumerated value that explains how ASP.NET stores session-state information. This storage mode is determined based on the web.config settings discussed in the “Configuring Session State” section later in this chapter
SessionID	Provides a string with the unique session identifier for the current client
Timeout	Determines the number of minutes that will elapse before the current session is abandoned, provided that no more requests are received from the client. This value can be changed programmatically, letting you make the session collection longer when needed.
Abandon()	Cancels the current session immediately and releases all the memory it occupied. This is a useful technique in a logoff page to ensure that server memory is reclaimed as quickly as possible.
Clear()	Removes all the session items but doesn’t change the current session identifier.

Table 6.1: HttpSessionState Members
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

6.6.4 A Session-State Example

```
public class Furniture
{
    public string Name;
    public string Description;
    public decimal Cost;
      public Furniture(string name, string description, decimal cost)
      {
         Name = name;
         Description = description;
         Cost = cost;
       }
}
```

- Three Furniture objects are created the first time the page is loaded, and they're stored in session state.
- The user can then choose from a list of furniture-piece names.
- When a selection is made, the corresponding object will be retrieved, and its information will be displayed.

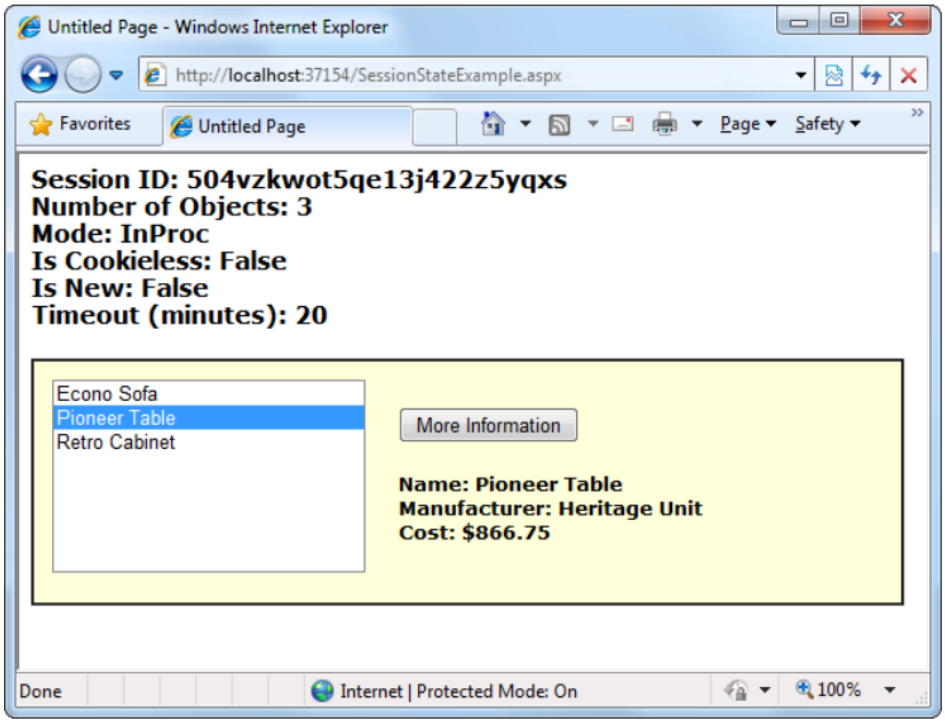


Figure 6.5: A session-state example with data objects
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

```
public partial class SessionStateExample : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
        {
            if (!this.IsPostBack)
            {
                // Create Furniture objects.
                Furniture piece1 = new Furniture("Econo Sofa", "Acme Inc.", 74.99M);
                Furniture piece2 = new Furniture("Pioneer Table", "Heritage Unit", 866.75M);
                Furniture piece3 = new Furniture("Retro Cabinet", "Sixties Ltd.", 300.11M);
                // Add objects to session state.
```

```
            Session["Furniture1"] = piece1;
            Session["Furniture2"] = piece2;
            Session["Furniture3"] = piece3;
            // Add rows to list control.
            lstItems.Items.Add(piece1.Name);
            lstItems.Items.Add(piece2.Name);
            lstItems.Items.Add(piece3.Name);
        }
        // Display some basic information about the session.
        // This is useful for testing configuration settings.
        lblSession.Text = "Session ID: " + Session.SessionID;
    lblSession.Text+="<br/>Number of Objects: ";
        lblSession.Text += Session.Count.ToString();
        lblSession.Text+="<br/>Mode: " +
Session.Mode.ToString();
        lblSession.Text+="<br/>Is Cookieless: ";
        lblSession.Text += Session.IsCookieless.ToString();
        lblSession.Text+="<br/>Is New: ";
        lblSession.Text += Session.IsNewSession.ToString();
        lblSession.Text+="<br/>Timeout (minutes): ";
        lblSession.Text += Session.Timeout.ToString();
    }
    protected void cmdMoreInfo_Click(Object sender, EventArgs e)
    {
        if (lstItems.SelectedIndex == -1)
        {
            lblRecord.Text = "No item selected.";
        }
        else
        {
            // Construct the right key name based on the index.
            string key = "Furniture" + (lstItems.SelectedIndex +
1).ToString();
            // Retrieve the Furniture object from session state.
            Furniture piece = (Furniture)Session[key];

            // Display the information for this object.
```

```
            lblRecord.Text = "Name: " + piece.Name;
            lblRecord.Text+="<br/>Manufacturer: ";
            lblRecord.Text += piece.Description;
            lblRecord.Text+="Cost: " + piece.Cost.ToString("c");
        }
      }
    }
```

6.7 CONFIGURING SESSION STATE

- You configure session state through the web.config file for your current application.
- The configuration file allows you to set advanced options such as the timeout and the session-state mode.
- The following listing shows the most important options that you can set for the element.

```
<configuration>
  ...
   <system.web>
    ...
        <sessionState
     cookieless="UseCookies"
     cookieName="ASP.NET_SessionId"
         regenerateExpiredSessionId="false"
         timeout="20"
         mode="InProc"
         stateConnectionString="tcpip=127.0.0.1:42424"
         stateNetworkTimeout="10"
         sqlConnectionString="data source=127.0.0.1;Integrated
        Security=SSPI"
         sqlCommandTimeout="30"
         allowCustomSqlDatabase="false"
         customProvider=""
         compressionEnabled="false"
      />
    </system.web>
</configuration>
```

6.8 APPLICATION STATE

- Application state allows you to store global objects that can be accessed by any client.
- Application state is based on the System.Web.HttpApplicationState class, which is provided in all web pages through the built-in Application object.
- Application state is similar to session state.
- It supports the same type of objects, retains information on the server, and uses the same dictionary-based syntax.

Example :

```
protected void Page_Load(Object sender, EventArgs e)
 {
     // Retrieve the current counter value.
     int count = 0;
     if (Application["HitCounterForOrderPage"] != null)
     {
       count = (int)Application["HitCounterForOrderPage"];
     }
     // Increment the counter.
     count++;
     // Store the current counter value.
     Application["HitCounterForOrderPage"] = count;
     lblCounter.Text = count.ToString();
}
```

- Once again, application-state items are stored as objects, so you need to cast them when you retrieve them from the collection.
- Items in application state never time out.
- They last until the application or server is restarted or the application domain refreshes itself

6.9 SUMMARY

- State management is the art of retaining information between requests.
- Usually, this information is user-specific (such as a list of items in a shopping cart, a username, or an access level), but sometimes it's global to the whole application (such as usage statistics that track site activity).

- Because ASP.NET uses a disconnected architecture, you need to explicitly store and retrieve state information with each request.
- The approach you choose to store this data affects the performance, scalability, and security of your application

6.10 REFERANCE

- Beginning ASP.NET in C# by Matthew MacDonald

6.11 QUESTIONS

1. Write and explain ViewState.
2. What is Cross-page posting.
3. What is Query string.
4. What is Cookies explain with example.
5. Explain Application state.

❄❄❄❄❄❄❄

7

VALIDATION

Unit Structure :

7.0 OBJECTIVES

- This chapter presents some of the most useful controls that are included in ASP.NET: the validation controls.
- These controls take a potentially time-consuming and complicated task—verifying user input and reporting errors—and automate it.
- Each validation control, or validator, has its own built-in logic.
- Some check for missing data, others verify that numbers fall in a predefined range, and so on. In many cases, the validation controls allow you to verify user input without writing a line of code.

7.1 INTRODUCTION

- As any seasoned developer knows, the people using your website will occasionally make mistakes. What's particularly daunting is the range of possible mistakes that users can make.
- Here are some common examples:
  - A user might ignore an important field and leave it blank. • If you disallow blank values, a user might type in semi-random nonsense to circumvent your checks.
  - A user might make an honest mistake, such as entering a typing error, entering a nonnumeric character in a number field, or submitting the wrong type of information

- A malicious user might try to exploit a weakness in your code by entering carefully structured wrong values. For example, an attacker might attempt to cause a specific error that will reveal sensitive information.

- ASP.NET aims to save you this trouble and provide you with a reusable framework of validation controls that manages validation details by checking fields and reporting on errors automatically.

- These controls can even use client-side JavaScript to provide a more dynamic and responsive interface while still providing ordinary validation for older browsers

7.2 VALIDATION CONTROLS

- ASP.NET provides five validator controls, which are described in Table.

- Four are targeted at specific types of validation, while the fifth allows you to apply custom validation routines

Control Class	Description
RequiredFieldValidator	Validation succeeds as long as the input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression
CustomValidator	Validation is performed by a user-defined function.

Table 7.1: Validator Controls
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

7.3 SERVER-SIDE VALIDATION

- If we have a client-side validation, after execution of client-side validation this validation is get executed.

- Server-side validation compulsory because a user or hacker can send the data through different channels.
- Server-side validation is done after the user submits the data to server or data post back to server.
- Server-side validation uses several languages like ASP.Net, PHP etc.
- After finishing validation process on the Server Side, the feedback is sent to the client by server using a new dynamically created web page.
- Its better to validate user input on Server Side because we need to protect the data from unauthorised users or hackers.
- It is more secure than client-side validation.

7.4 CLIENT-SIDE VALIDATION

- In modern browsers, ASP.NET automatically adds JavaScript code for client-side validation.
- In this case, when the user clicks a CausesValidation button, the same error messages will appear without the page needing to be submitted and returned from the server.
- This increases the responsiveness of your web page.
- However, even if the page validates successfully on the client side, ASP.NET still revalidates it when it's received at the server.
- This is because it's easy for an experienced user to manipulate client-side validation.
- For example, a malicious user might delete the block of JavaScript validation code and continue working with the page.
- By performing the validation at both ends, ASP.NET makes sure your application can be as responsive as possible while also remaining secure.

7.5 HTML5 VALIDATION

- HTML language, adds new client-side validation features that can help catch errors.
- The problem is that HTML5 validation is inconsistent—it works differently in different browsers, and many browsers offer only partial support.

7.5.1 The Validation Controls

- The validation controls are found in the System.Web.UI.WebControls namespace and inherit from the BaseValidator class.

- This class defines the basic functionality for a validation control.

Properties of the BaseValidator Class

Property	Description
ControlToValidate	Identifies the control that this validator will check. Each validator can verify the value in one input control.
ErrorMessage and ForeColor	If validation fails, the validator control can display a text message
Display	Allows you to configure whether this error message will be inserted into the page dynamically when it's needed or whether an appropriate space will be reserved for the message.
IsValid	After validation is performed, this returns true or false depending on whether it succeeded or failed
Enabled	When set to false, automatic validation will not be performed for this control when the page is submitted.
EnableClientScript	If set to true, ASP.NET will add JavaScript and DHTML code to allow client-side validation on browsers that support it.

Table 7.2: Properties of the BaseValidator Class
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Validator-Specific Properties

Validator	Added Members
RequiredFieldValidator	None required
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

Table 7.3: Validator-Specific Properties
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Validation Example

- This test uses a single Button web control, two TextBox controls, and a RangeValidator control that validates the first text box.
- If validation fails, the RangeValidator control displays an error message, so you should place this control immediately next to the TextBox it's validating.
- The second text box does not use any validation.

```
<asp:TextBox id="txtValidated" runat="server" />
<asp:RangeValidator id="RangeValidator" runat="server"
 ErrorMessage="This Number Is Not In The Range"
 ControlToValidate="txtValidated"
 MaximumValue="10" MinimumValue="1"
 ForeColor="Red" Font-Bold="true"
 Type="Integer" />
<br /><br />
Not validated:
<asp:TextBox id="txtNotValidated" runat="server" /><br /><br />
<asp:Button id="cmdOK" runat="server" Text="OK"
OnClick="cmdOK_Click" />
<br /><br />
<asp:Label id="lblMessage" runat="server"
 EnableViewState="False" />
```

Finally, here is the code that responds to the button click:

```
protected void cmdOK_Click(Object sender, EventArgs e)
{
 lblMessage.Text = "cmdOK_Click event handler executed.";
}
```

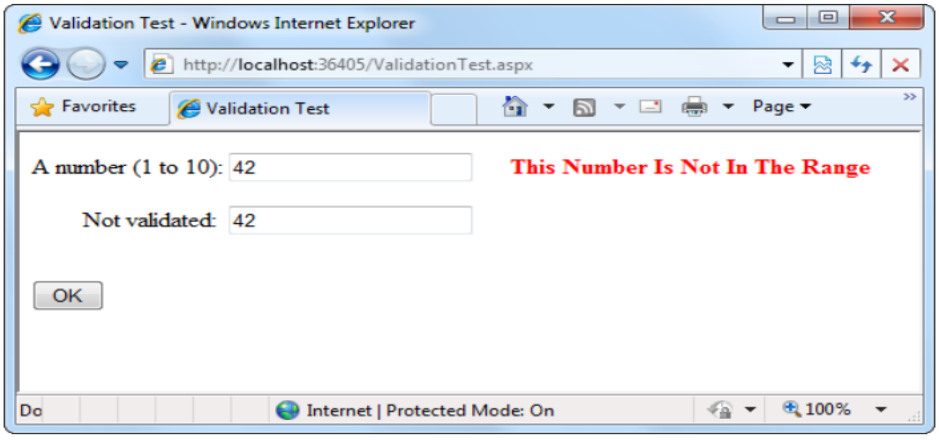


Figure 7.1: Failed validation
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

7.6 MANUAL VALIDATION

- You can create manual validation in one of three ways:

  1. Use your own code to verify values. In this case, you won't use any of the ASP.NET validation controls.

  2. Disable the EnableClientScript property for each validation control. This allows an invalid page to be submitted, after which you can decide what to do with it depending on the problems that may exist.

  3. Add a button with CausesValidation set to false. When this button is clicked, manually validate the page by calling the Page.Validate() method. Then examine the IsValid property and decide what to do.

- The next example uses the second approach. After the page is submitted, it examines all the validation controls on the page by looping through the Page.Validators collection.

- This technique adds a feature that wouldn't be available with automatic validation, which uses the ErrorMessage property.

```
protected void cmdOK_Click(Object sender, EventArgs e)
{
    string errorMessage = "<b>Mistakes found:<b/>";
    // Search through the validation controls.
     foreach (BaseValidator ctrl in this.Validators)
     {
        if (!ctrl.IsValid)
        {
            errorMessage += ctrl.ErrorMessage +"<br/> ";
           // Find the corresponding input control, and change the
           // generic Control variable into a TextBox variable.
           // This allows access to the Text property.
           TextBox ctrlInput =
(TextBox)this.FindControl(ctrl.ControlToValidate);
           errorMessage += " * Problem is with this input: ";
           errorMessage += ctrlInput.Text + "<br/>";
        }
      }
lblMessage.Text = errorMessage;
}
```

7.7 VALIDATION WITH REGULAR EXPRESSIONS

- One of ASP.NET's most powerful validation controls is the RegularExpressionValidator, which validates text by determining whether it matches a specific pattern.
- For example, e-mail addresses, phone numbers, and file names are all examples of text that has specific constraints.
- A phone number must be a set number of digits, an e-mail address must include exactly one @ character (with text on either side), and a file name can't include certain special characters such as \ and ?.
- One way to define patterns like these is with regular expressions.

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7+8 matches 7778 but not 8.
( )	Groups a subexpression that will be treated as a single element. For example, (78)+ matches 78 and 787878.
{m,n}	The previous character (or subexpression) can occur from m to n times. For example, A{1,3} matches A, AA, or AAA.
|	Either of two matches. For example, 8|6 matches 8 or 6
[ ]	Matches one character in a range of valid characters. For example, [A-C] matches A, B, or C.
[^]	Matches one character in a range of valid characters. For example, [A-C] matches A, B, or C.
.	Any character except a newline. For example, .here matches where and there.
\s	Any whitespace character (such as a tab or space).
\S	Any nonwhitespace character
\d	Any digit character
\D	Any character that isn't a digit.
\w	Any "word" character (letter, number, or underscore).
\W	Any character that isn't a "word" character (letter, number, or underscore).

Table 7.4: Common (and useful) regular expressions.
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Commonly Used Regular Expressions

Content	Regular Expression	Description
E-mail address*	\S+@\S+\.\S+	Check for an at (@) sign and dot (.) and allow nonwhitespace characters only.
Password	\w+	Any sequence of one or more word characters (letter, space, or underscore).
Specific-length password	\w{4,10}	A password that must be at least four characters long but no longer than ten characters.
Advanced password	[a-zA-Z]\w{3,9}	As with the specific-length password, this regular expression will allow four to ten total characters. The twist is that the first character must fall in the range of a–z or A–Z
Another advanced password	[a-zA-Z]\w*\d+\w*	This password starts with a letter character, followed by zero or more word characters, one or more digits, and then zero or more word characters
Limited-length field	\S{4,10}	Like the password example, this allows four to ten characters, but it allows special characters
US Social Security number	\d{3}-\d{2}-\d{4}	A sequence of three, two, and then four digits, with each group separated by a dash. You could use a similar pattern when requiring a phone number.

Table 7.5: Commonly Used Regular Expressions
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Example :

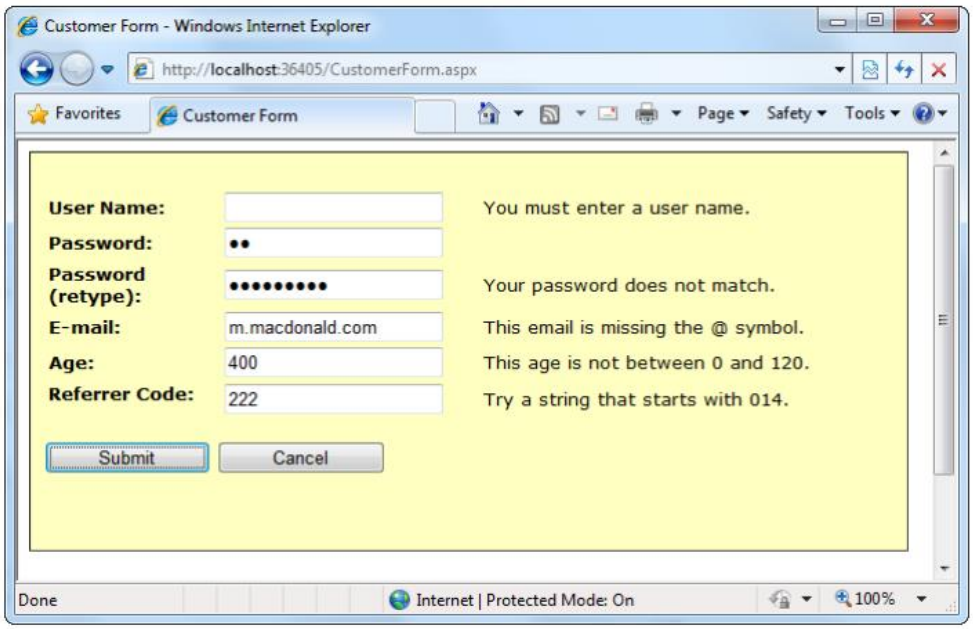


Figure 7.2: A sample customer form
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Several types of validation are taking place on the customer form:

- Three RequiredFieldValidator controls make sure the user enters a username, a password, and a password confirmation.
- A CompareValidator ensures that the two versions of the masked password match.
- A RegularExpressionValidator checks that the e-mail address contains an at (@) symbol.
- A RangeValidator ensures the age is a number from 0 to 120.
- A CustomValidator performs a special validation on the server of a "referrer code." This code verifies that the first three characters make up a number that is divisible by 7.

The tags for the validator controls are as follows:

```
<asp:RequiredFieldValidator id="vldUserName" runat="server"
 ErrorMessage="You must enter a user name."
 ControlToValidate="txtUserName" />
<asp:RequiredFieldValidator id="vldPassword" runat="server"
 ErrorMessage="You must enter a password."
 ControlToValidate="txtPassword" />
<asp:CompareValidator id="vldRetype" runat="server"
```

```
 ErrorMessage="Your password does not match."
 ControlToCompare="txtPassword" ControlToValidate="txtRetype" />
<asp:RequiredFieldValidator id="vldRetypeRequired" runat="server"
 ErrorMessage="You must confirm your password."
 ControlToValidate="txtRetype" />
<asp:RegularExpressionValidator id="vldEmail" runat="server"
 ErrorMessage="This email is missing the @ symbol."
 ValidationExpression=".+@.+" ControlToValidate="txtEmail" />
<asp:RangeValidator id="vldAge" runat="server"
 ErrorMessage="This age is not between 0 and 120." Type="Integer"
 MinimumValue="0" MaximumValue="120"
 ControlToValidate="txtAge" />
<asp:CustomValidator id="vldCode" runat="server"
 ErrorMessage="Try a string that starts with 014."
 ValidateEmptyText="False"
 OnServerValidate="vldCode_ServerValidate"
 ControlToValidate="txtCode" />
```

The form provides two validation buttons—one that requires validation and one that allows the user to cancel the task gracefully:

```
<asp:Button id="cmdSubmit" runat="server"
 OnClick="cmdSubmit_Click" Text="Submit"></asp:Button>
<asp:Button id="cmdCancel" runat="server"
 CausesValidation="False" OnClick="cmdCancel_Click" Text="Cancel">
</asp:Button>
```

Here’s the event-handling code for the buttons:

```
protected void cmdSubmit_Click(Object sender, EventArgs e)
{
 if (Page.IsValid)
 {
 lblMessage.Text = "This is a valid form.";
 }
```

```
}
protected void cmdCancel_Click(Object sender, EventArgs e)
{
 lblMessage.Text = "No attempt was made to validate this form.";
}
```

The only form-level code that is required for validation is the custom validation code. The validation takes place in the event handler for the CustomValidator.ServerValidate event.

```
protected void vldCode_ServerValidate(Object source,
ServerValidateEventArgs e)
{
 try
 {
 // Check whether the first three digits are divisible by seven.
 int val = Int32.Parse(e.Value.Substring(0, 3));
 if (val % 7 == 0)
 {
 e.IsValid = true;
 }
 else
{
 e.IsValid = false;
 }
 }
 catch
 {
 // An error occurred in the conversion.
 // The value is not valid.
 e.IsValid = false;
 }
}
```

7.8 SUMMARY

- In this chapter, you learned how to use one of ASP.NET's most practical features: validation.
- You saw how ASP. NET combines server-side and client-side validation to ensure bulletproof security without sacrificing the usability of your web pages.
- You also looked at the types of validation provided by the various validation controls, and even brushed up on the powerful pattern-matching syntax used for regular expressions

7.9 REFERANCE

- Beginning ASP.NET in C# by Matthew MacDonald

7.10 QUESTIONS

1. Write and explain Validation control.
2. Explain Server-side validation
3. Explain Client-side validation
4. Explain Manual validation
5. Explain validation with Regular expression.

❄❄❄❄❄❄❄

8

RICH CONTROLS

Unit Structure :

8.0 OBJECTIVES

- Rich controls are web controls that model complex user interface elements.
- Although no strict definition exists for what is and what isn't a rich control, the term commonly describes a web control that has an object model that's distinctly separate from the HTML it generates.

8.1 INTRODUCTION

- Rich control provides object model that has complex HTML representation and also client-side JavaScript
- Rich controls are web controls that model complex user interface elements**.**
- A typical rich control can be programmed as a single object but renders itself using a complex sequence of HTML elements.
- Rich controls can also react to user actions (such as a mouse click on a specific region of the control) and raise more-meaningful events that your code can respond to on the web server.
- In other words, rich controls give you a way to create advanced user interfaces in your web pages without writing lines of convoluted HTML.

8.2 CALENDAR CONTROL

- The Calendar control presents a miniature calendar that you can place in any web page.

```
<asp:Calendar id="MyCalendar" runat="server" />
```

- The Calendar control presents a single-month view
- The user can navigate from month to month by using the navigational arrows, at which point the page is posted back and ASP.NET automatically provides a new page with the correct month values

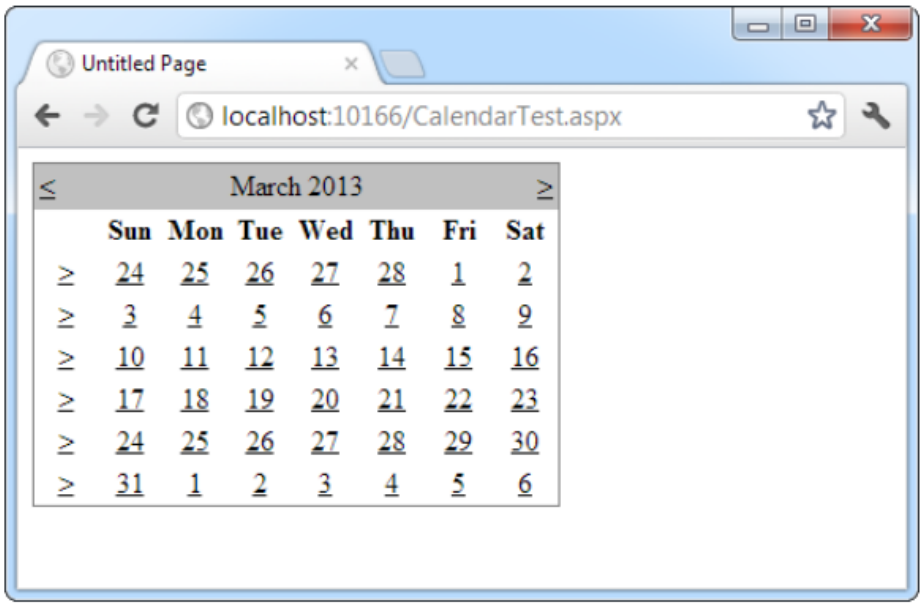


Figure 8.1: The default Calendar
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Depending on the value you choose, you can allow users to select days (Day), entire weeks (DayWeek), whole months (DayWeekMonth), or render the control as a static calendar that doesn't allow selection (None).
- You may also want to set the Calendar.FirstDayOfWeek property to configure how a week is shown.
- For example, set FirstDayOfWeek to the enumerated value Sunday, and weeks will be selected from Sunday to Saturday.
- The following code demonstrates this technique:

```
lblDates.Text = "You selected these dates:<br />";
foreach (DateTime dt in MyCalendar.SelectedDates)
{
 lblDates.Text += dt.ToLongDateString() + "<br />";
}
```

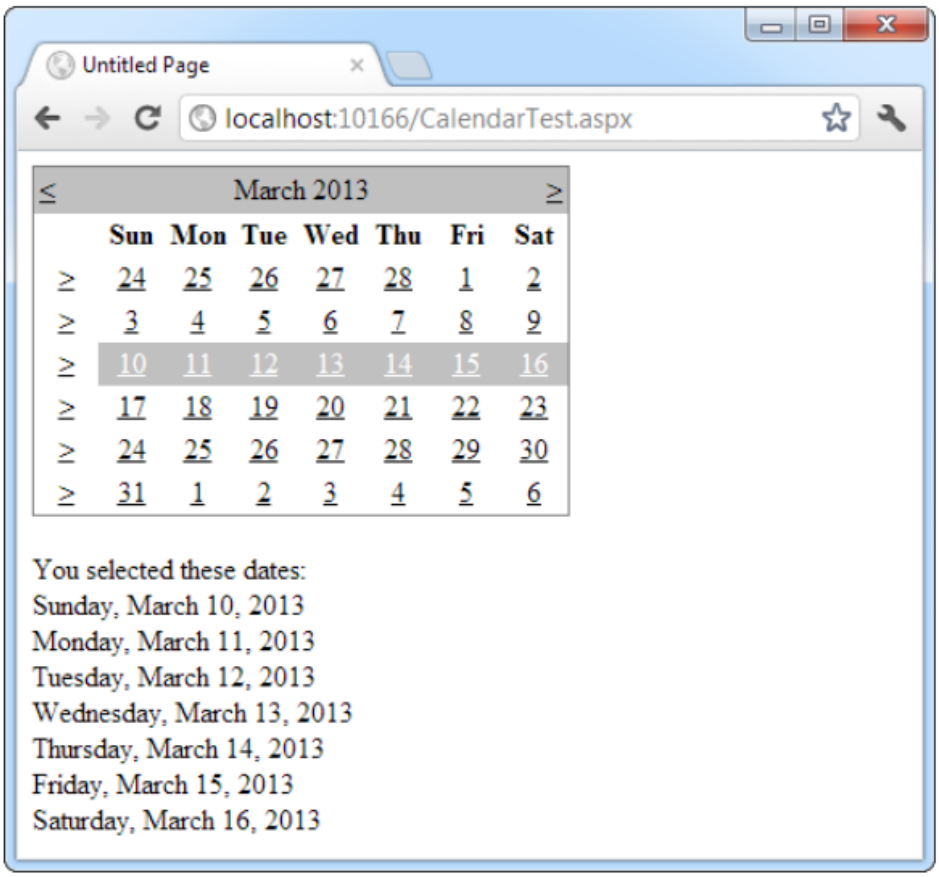


Figure 8.2: Selecting multiple dates
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.2.1 Formatting the Calendar

- The Calendar control provides a whole host of formatting-related properties.

Member	Description
DayHeaderStyle	The style for the section of the Calendar that displays the days of the week (as column headers).
DayStyle	The default style for the dates in the current month
NextPrevStyle	The style for the navigation controls in the title section that move from month to month.
OtherMonthDayStyle	The style for the dates that aren't in the currently displayed month. These dates are used to "fill in" the calendar grid. For example, the first few cells in the topmost row may display the last few days from the previous month.

Member	Description
SelectedDayStyle	The style for the selected dates on the calendar
SelectorStyle	The style for the week and month date selection controls.
TitleStyle	The style for the title section.
TodayDayStyle	The style for the date designated as today (represented by the TodaysDate property of the Calendar control).
WeekendDayStyle	The style for dates that fall on the weekend.

Table 8.1: Properties for Calendar Styles
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- You can adjust each style by using the Properties window.
- For a quick shortcut, you can set an entire related color scheme by using the Calendar's Auto Format feature.
- To do so, start by selecting the Calendar on the design surface of a web form. Then click the arrow icon that appears next to its top-right corner to show the Calendar's smart tag, and click the Auto Format link.
- You'll be presented with a list of predefined formats that set the style properties

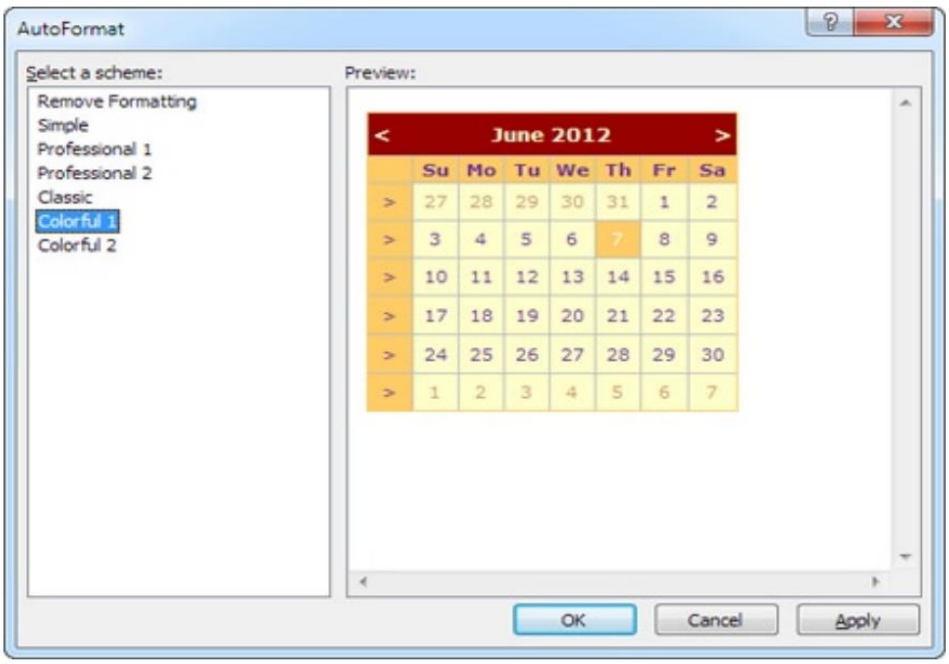


Figure 8.3: Calendar Styles
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.2.2 CalendarDay Properties

Property	Description
Date	The DateTime object that represents this date.
IsWeekend	True if this date falls on a Saturday or Sunday.
IsToday	True if this value matches the Calendar.TodaysDate property, which is set to the current day by default.
IsOtherMonth	True if this date doesn't belong to the current month but is displayed to fill in the first or last row.
IsSelectable	Allows you to configure whether the user can select this day.

Table 8.2: CalendarDay Properties
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Members of the Calendar control class.

Member	Description
Caption and CaptionAlign	Gives you an easy way to add a title to the calendar. By default, the caption appears at the top of the title area, just above the month heading.
CellPadding	ASP.NET creates a date in a separate cell of an invisible table. CellPadding is the space, in pixels, between the border of each cell and its contents.
CellSpacing	The space, in pixels, between cells in the same table.
DayNameFormat	Determines how days are displayed in the calendar header. Valid values are Full (as in Sunday), FirstLetter (S), FirstTwoLetters (Su), and Short (Sun), which is the default.
FirstDayOfWeek	Determines which day is displayed in the first column of the calendar. The values are any day name from the FirstDayOfWeek enumeration (such as Sunday). By default, this is Sunday.
TitleFormat	Configures how the month is displayed in the title area. Valid values include Month and MonthYear
TodaysDate	Sets which day should be recognized as the current date and formatted with the TodayDayStyle. This defaults to the current day on the web server.

Member	Description
VisibleDate	Gets or sets the date that specifies what month will be displayed in the calendar. This allows you to change the calendar display without modifying the current date selection.
ShowDayHeader, ShowGridLines, ShowNextPrevMonth, and ShowTitle	These Boolean properties allow you to configure whether various parts of the calendar are shown, including the day titles, gridlines between every day, the previous/next month navigation links, and the title section. Note that hiding the title section also hides the next and previous month navigation controls.

Table 8.3: Calendar Members
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.3 ADROTATOR CONTROL

- The basic purpose of the AdRotator is to provide a graphic on a page that is chosen randomly from a group of possible images.
- In other words, every time the page is requested, an image is selected at random and displayed, which is the rotation indicated by the name AdRotator.
- One use of the AdRotator is to show banner-style advertisements on a page, but you can use it anytime you want to vary an image randomly.

8.3.1 The Advertisement File

- The AdRotator stores its list of image files in an XML file. This file uses the format shown here:

```
<Advertisements>
 <Ad>
    <ImageUrl>prosetech.jpg</ImageUrl>
    <NavigateUrl>http://www.prosetech.com</NavigateUrl>
    <AlternateText>ProseTech Site</AlternateText>
    <Impressions>1</Impressions>
    <Keyword>Computer</Keyword>
 </Ad>
</Advertisements>
```

- This example shows a single possible advertisement, which the AdRotator control picks at random from the list of advertisements.

- To add more advertisements, you would create multiple elements and place them all inside the root element:

```
<Advertisements>
    <Ad>
    <!-- First ad here. -->
    </Ad>
 <Ad>
    <!-- Second ad here. -->
    </Ad>
</Advertisements>
```

- Each element has a number of other important properties that configure the link, the image, and the frequency.

Element	Description
ImageUrl	The image that will be displayed. This can be a relative link (a file in the current directory) or a fully qualified Internet URL.
NavigateUrl	The link that will be followed if the user clicks the banner. This can be a relative or fully qualified URL.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed. This text will also be used as a tooltip in some newer browsers.
Impressions	A number that sets how often an advertisement will appear. This number is relative to the numbers specified for other ads. For example, a banner with the value 10 will be shown twice as often (on average) as the banner with the value 5.
Keyword	A keyword that identifies a group of advertisements. You can use this for filtering. For example, you could create ten advertisements and give half of them the keyword Retail and the other half the keyword Computer.

Table 8.4: Advertisement File Elements
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.3.2 The AdRotator Class

- The actual AdRotator class provides a limited set of properties.
- You specify both the appropriate advertisement file in the AdvertisementFile property and the type of window that the link should follow (the Target window). The target can name a specific frame

Target	Description
_blank	The link opens a new unframed window.
_parent	The link opens in the parent of the current frame.
_self	The link opens in the current frame.
_top	The link opens in the topmost frame of the current window

Table 8.5: Advertisement File Elements
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Optionally, you can set the KeywordFilter property so that the banner will be chosen from a specific keyword group. This is a fully configured AdRotator tag:

```
<asp:AdRotator ID="Ads" runat="server" AdvertisementFile="MainAds.xml"
 Target=”_blank” KeywordFilter=”Computer” />
```

- The event-handling code for this example simply configures a HyperLink control named lnkBanner based on the randomly selected advertisement:

```
protected void Ads_AdCreated(Object sender, AdCreatedEventArgs e)
{
 // Synchronize the Hyperlink control.
 lnkBanner.NavigateUrl = e.NavigateUrl;
 // Syncrhonize the text of the link.
 lnkBanner.Text = “Click here for information about our sponsor: “;
 lnkBanner.Text += e.AlternateText;
}
```

8.4 MULTIVIEW CONTROL

- The MultiView is the simpler of the two multiple-view controls.
- The MultiView gives you a way to declare multiple views and show only one at a time.
- Creating a MultiView is suitably straightforward. You add the tag to your .aspx page file and then add one tag inside it for each separate view:

```
<asp:MultiView ID="MultiView1" runat="server">
    <asp:View ID=”View1” runat=”server”>...</asp:View>
    <asp:View ID=”View2” runat=”server”>...</asp:View>
    <asp:View ID=”View3” runat=”server”>...</asp:View>
</asp:MultiView>
```

8.4.1 Creating Views

- Full markup for a MultiView that splits the greeting card controls into three views named View1, View2, and View3:

```
<asp:MultiView ID="MultiView1" runat="server" >
<asp:View ID=”View1” runat=”server”>
  Choose a foreground (text) color:<br />
  <asp:DropDownList ID=”lstForeColor” runat=”server” AutoPostBack=”True”
  OnSelectedIndexChanged=”ControlChanged” />
  <br /><br />
  Choose a background color:<br />
  <asp:DropDownList ID=”lstBackColor” runat=”server” AutoPostBack=”True”
  OnSelectedIndexChanged=”ControlChanged” />
</asp:View>
<asp:View ID=”View2” runat=”server”>
  Choose a border style:<br />
  <asp:RadioButtonList ID=”lstBorder” runat=”server” AutoPostBack=”True”
  OnSelectedIndexChanged=”ControlChanged” RepeatColumns=”2” />
  <br />
  <asp:CheckBox ID=”chkPicture” runat=”server” AutoPostBack=”True”
  OnCheckedChanged=”ControlChanged” Text=”Add the Default Picture” />
</asp:View>

<asp:View ID=”View3” runat=”server”>
  Choose a font name:<br />
  <asp:DropDownList ID=”lstFontName” runat=”server” AutoPostBack=”True”
  OnSelectedIndexChanged=”ControlChanged” />
  <br /><br />
  Specify a font size:<br />
  <asp:TextBox ID=”txtFontSize” runat=”server” AutoPostBack=”True”
  OnTextChanged=”ControlChanged” />
  <br /><br />
  Enter the greeting text below:<br />
  <asp:TextBox ID=”txtGreeting” runat=”server” AutoPostBack=”True”
  OnTextChanged=”ControlChanged” TextMode=”MultiLine” />
</asp:View>
</asp:MultiView>
```

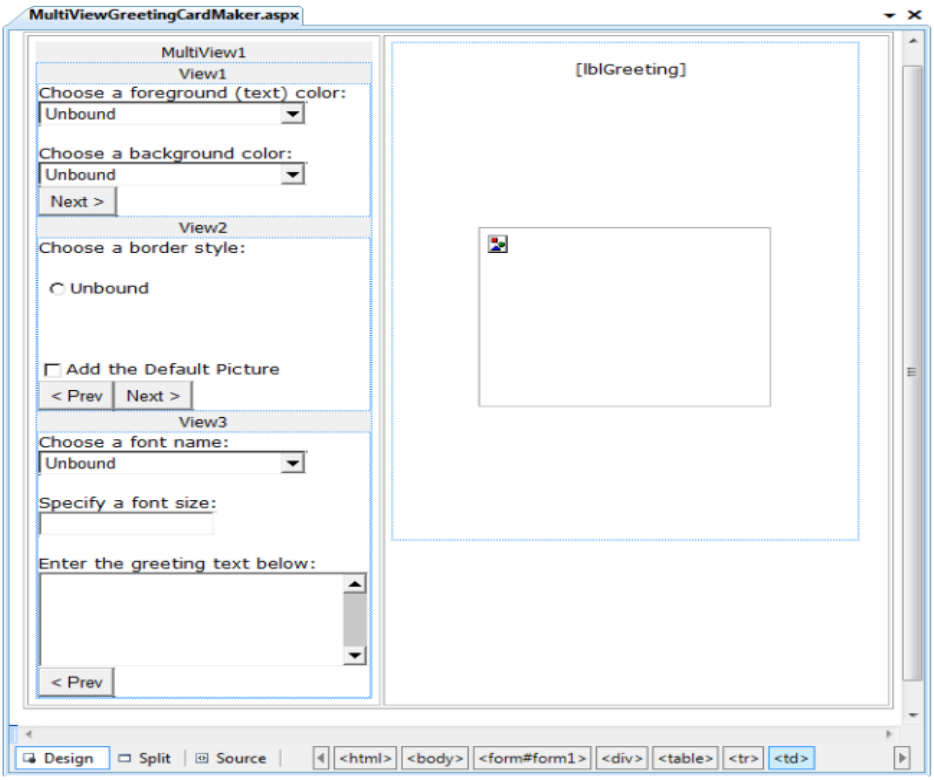


Figure 8.4: Designing multiple views
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.4.2 Showing a View

- If you run this example, you won't see what you expect.
- The MultiView will appear empty on the page, and all the controls in all your views will be hidden.
- The reason this happens is that the MultiView.ActiveViewIndex property is, by default, set to –1. The ActiveViewIndex property determines which view will be shown.
- If you set the ActiveViewIndex to 0, however, you'll see the first view. Similarly, you can set it to 1 to show the second view, and so on.
- You can set this property by using the Properties window or using code:

  ```
  // Show the first view.

  MultiView1.ActiveViewIndex = 0;
  ```

- This example shows the first view (View1) and hides whatever view is currently being displayed, if any.

- You can also use the SetActiveView() method, which accepts any one of the view objects you've created, rather than the view name.

  MultiView1.SetActiveView(View1);

- Following table shows lists all the recognized command names. Each command name also has a corresponding static field in the MultiView class.

Command Name	MultiView Field	Description
PrevView	PreviousViewCommandName	Moves to the previous view.
NextView	NextViewCommandName	Moves to the next view.
SwitchViewByID	SwitchViewByIDCommandName	Moves to the view with a specific ID (string name). The ID is taken from the CommandArgument property of the button control.
SwitchViewByIndex	SwitchViewByIndexCommandName	Moves to the view with a specific numeric index. The index is taken from the CommandArgument property of the button control.

Table 8.6: Recognized Command Names for the MultiView
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

8.5 SUMMARY

- This chapter showed you how the rich Calendar, AdRotator, MultiView, and Wizard controls can go far beyond the limitations of ordinary HTML elements.
- When you're working with these controls, you don't need to think about HTML at all. Instead, you can focus on the object model that's defined by the control.

8.6 REFERENCE

- Beginning ASP.NET in C# by Matthew MacDonald

8.7 QUESTIONS

1. Explain Calendar control.
2. Write and explain formatting properties for calendar.
3. Write and explain members of the calender control.
4. Explain AdRotator control.
5. Explain multiview control.

❄❄❄❄❄❄❄

THEMES AND MASTER PAGES

Unit Structure:

9.0 OBJECTIVES

- Using the techniques, you've learned so far, you can create polished web pages and let users surf from one page to another.
- However, to integrate your web pages into a unified, consistent website, you need a few more tools.
- In this chapter, you'll consider three of the most important tools that you can use: styles, themes, and master pages.

9.1 INTRODUCTION

- ASP. NET includes themes feature, which plays a similar role as styles but works exclusively with server controls.
- Best feature for standardizing websites is master pages. Essentially, a master page is a blueprint for part of your website.
- Using a master page, you can define web page layout, complete with the usual details such as headers, menu bars, and ad banners.
- Once you've perfected a master page, you can use it to create content pages. Each content page automatically acquires the layout and the content of the linked master page.
- By using themes, and master pages, you can ensure that all the pages on your website share a standardized look and layout.

9.2 HOW THEMES WORK

- All themes are application specific.
- To use a theme in a web application, you need to create a folder that defines it.
- This folder needs to be placed in the App_Themes folder, which must be placed inside the top-level directory for your web application.
- To use theme, you need to create at least one skin file in the theme folder. A skin file is a text file with the .skin extension

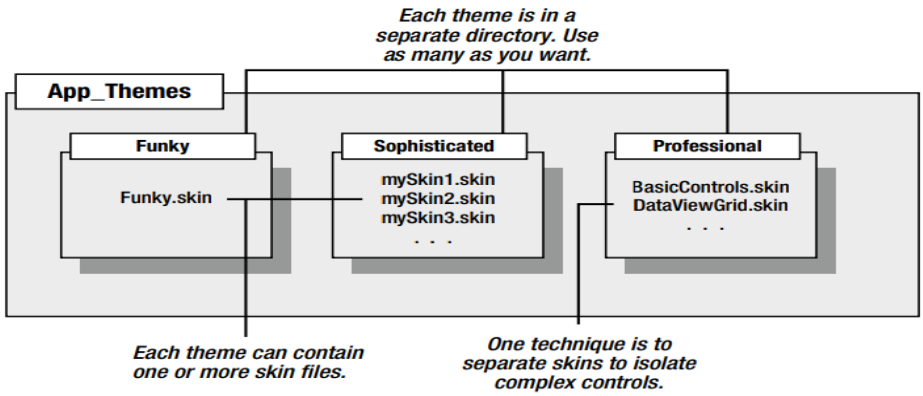


Figure 9.1: Themes and skins
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

9.3 APPLYING A SIMPLE THEME:

- To add a theme to your project, select Website➤ Add New Item, and choose Skin File.
- Visual Studio will warn you that skin files need to be placed in a subfolder of the App_Themes folder and ask you whether that's what you intended.
- If you choose Yes, Visual Studio will create a folder with the same name as your theme file. You can then rename the folder and the file to whatever you'd like to use.

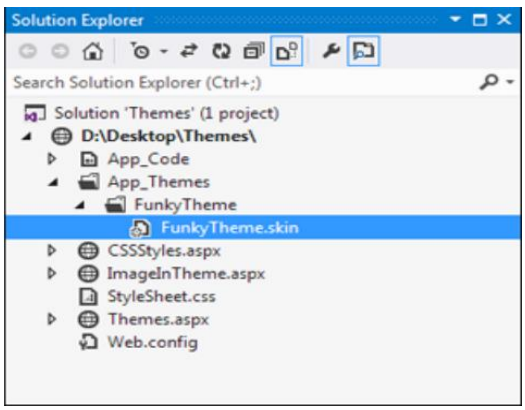


Figure 9.2: A theme in the Solution Explorer *(Ref: Beginning ASP.NET in C# by Matthew MacDonald)*

- Here's a sample skin file that sets background and foreground colors for several common controls:

```
<asp:ListBox runat="server" ForeColor ="White" BackColor="Orange"/>

<asp:TextBox runat="server" ForeColor ="White" BackColor="Orange"/>

<asp:Button runat="server" ForeColor ="White" BackColor ="Orange"/>
```

- To apply the theme in a web page, you need to set the Theme attribute of the Page directive to the folder name for your theme.

```
<%@ Page Language="C#" AutoEventWireup ="true" ... Theme="FunkyTheme" %>
```

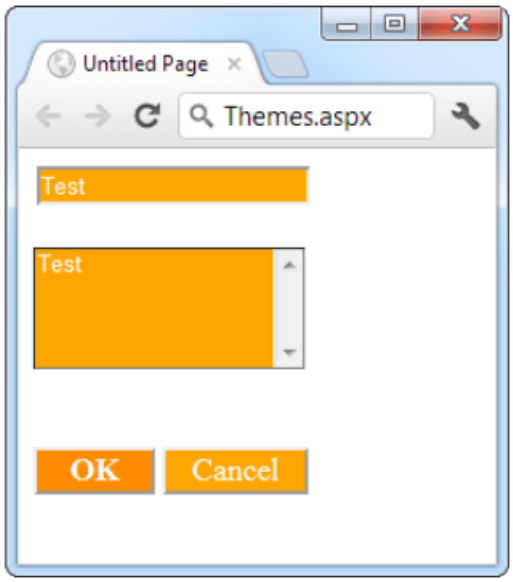


Figure 9.3: A simple page after theming
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

9.4 HANDLING THEME CONFLICTS:

- When properties conflict between your controls and your theme, the theme wins.
- However, in some cases you might want to change this behavior so that your controls can fine-tune a theme by specifically overriding certain details.
- ASP.NET gives you this option, but it's an all-or-nothing setting that applies to all the controls on the entire page.
- To make this change, just use the StyleSheetTheme attribute instead of the Theme attribute in the Page directive.

```
<%@ Page Language="C#" AutoEventWireup ="true" ... StyleSheetTheme="FunkyTheme" %>
```

- Now the custom yellow background of the ListBox control takes precedence over the background color specified by the theme.
- Figure shows the result—and a potential problem.
- Because the foreground color has been changed to white, the lettering in the large text box is now impossible to see. Overlapping formatting specifications can cause glitches like this, which is why it's often better to let your themes take complete control by using the Theme attribute.

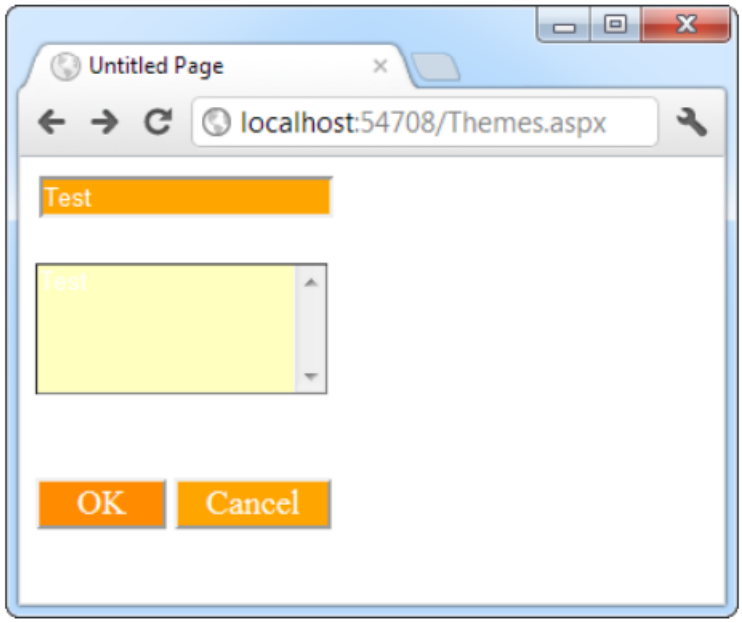


Figure 9.4: Giving the control tag precedence over the theme
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

9.5 SIMPLE MASTER PAGE AND CONTENT PAGE:

- Master pages are similar to ordinary ASP.NET pages. Like ordinary pages, master pages are text files that can contain HTML, web controls, and code.
- However, master pages have a different file extension (.master instead of .aspx), and they can't be viewed directly by a browser.
- Instead, master pages must be used by other pages, which are known as content pages. Essentially, the master page defines the page structure and the common ingredients.
- A single master page might define the layout for the entire site.
- Every page would use that master page, and as a result, every page would have the same basic organization and the same title, footer, and so on.

- However, each page would also insert its specific information, such as product descriptions, book reviews, or search results, into this template
- To create a master page in Visual Studio, select Website➤Add New Item from the menu. Select Master Page, give it a file name (such as SiteTemplate.master, used in the next example), and click Add.
- The ContentPlaceHolder is the portion of the master page that a content page can change. Or, to look at it another way, everything else that's set in the master page is unchangeable in a content page.
- If you add a header, that header appears in every content page.
- If you want to give the content page the opportunity to supply content in a specific section of the page, you need to add a ContentPlaceHolder.

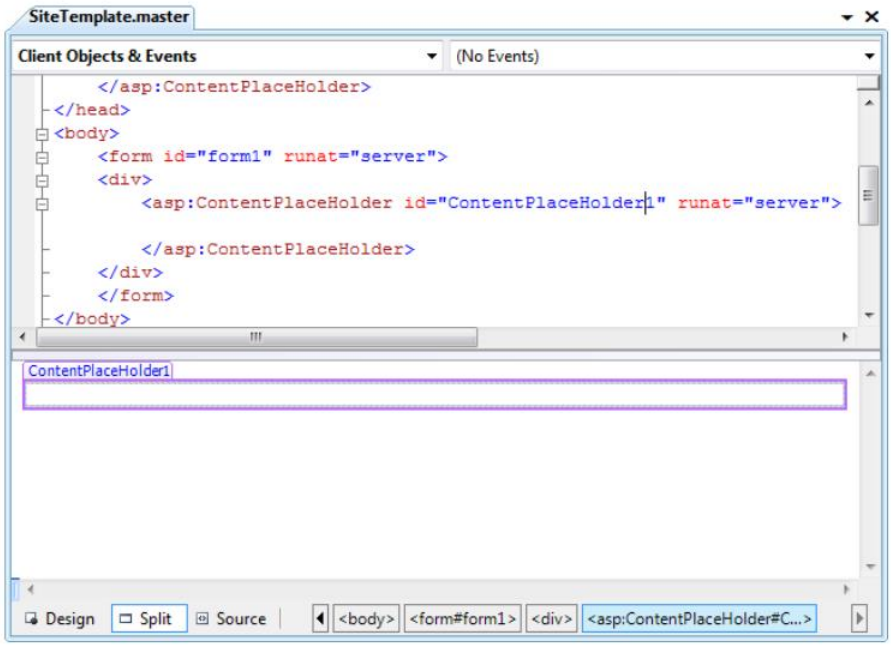


Figure 9.5: A new master page
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- When you first create a master page, you'll start with two ContentPlaceHolder controls.
- One is defined in thesection, which gives content pages the add page metadata, such as search keywords and style sheet links.
- The second, more important ContentPlaceHolder is defined in the section, and represents the displayed content of the page.
- It appears on the page as a faintly outlined box. If you click inside it or hover over it, the name of ContentPlaceHolder appears in a tooltip.

- Now you're ready to create a content page based on this master page. To take this step, select Website➤ Add New Item from the menu.
- Select Web Form, and choose to select a master page.
- Click Add. When you're prompted to choose a master page, use the one you created with the header and footer.

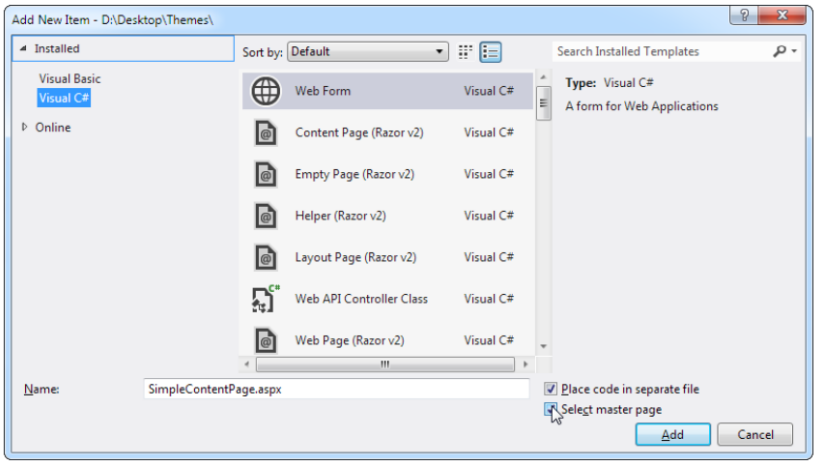


Figure 9.6: Creating a content page
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Your content page will have all the elements of the master page

9.6 CONNECTING MASTER PAGES AND CONTENT PAGES:

- When you create a master page, you're building something that looks much like an ordinary ASP.NET web form.
- The key difference is that, although web forms start with the Page directive, a master page starts with a Master directive that specifies the same information.
- Here's the Master directive for the simple master page shown in the previous example:

```
<%@ Master Language="C#" AutoEventWireup ="true"
CodeFile="SiteTemplate.master.cs"
Inherits="SiteTemplate" %>
```

- The ContentPlaceHolder is less interesting. You declare it like any ordinary control. Here's the complete code for the simple master page:

```
<%@ Master Language="C#" AutoEventWireup ="true"
CodeFile="SiteTemplate.master.cs"
 Inherits="SiteTemplate" %>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
 <form id="form1" runat="server">
 <img src ="apress.jpg" /><br />
 <asp:ContentPlaceHolder id="ContentPlaceHolder1"
runat="server">
 </asp:ContentPlaceHolder>
 <i>This is a simple footer.</i>
 </form>
</body>
</html>
```

- When you create a content page, ASP.NET links your page to the master page by adding an attribute to the Page directive.
- This attribute, named MasterPageFile, indicates the associated master page. Here's what it looks like:

```
<%@ Page Language="C#" MasterPageFile
="~/SiteTemplate.master" AutoEventWireup="true"
CodeFile="SimpleContentPage.aspx.cs"
 Inherits="SimpleContentPage" Title="Untitled Page" %>
```

- Notice that the MasterPageFile attribute begins with the path ~/ to specify the root website folder.

9.7 MASTER PAGE WITH MULTIPLE CONTENT REGIONS

- Master pages aren't limited to one ContentPlaceHolder. Instead, you can insert as many as you need to give the client the ability to intersperse content in various places. All you need to do is add multiple ContentPlaceHolder controls and arrange them appropriately.
- Example:

```
<%@ Master Language="C#" AutoEventWireup ="true"
 CodeFile="MultipleContent.master.cs" Inherits="MultipleContent"
%>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title>Untitled Page</title>
```

```
        </head>
    <body>
        <form id="form1" runat="server">
            <img src ="apress.jpg" /><br />
            <asp:ContentPlaceHolder id="MainContent"
            runat="server">
            </asp:ContentPlaceHolder>
            <i>
                <div style="...">
                    <b>OTHER LINKS</b>
                    <br />
                    <asp:ContentPlaceHolder id="OtherLinksContent"
                    runat="server">
                    </asp:ContentPlaceHolder>
                </div>
                This is a simple footer.
            </i>
        </form>
    </body>
    </html>
```

- When you create a new content page based on this master page, Visual Studio will start you with one Content control for each ContentPlaceHolder in the master page, making your life easy.
- All you need to do is insert the appropriate information.
- Here's a slightly shortened example, with some of the text replaced with an ellipsis (. . .) to save space:

```
<%@ Page Language="C#" MasterPageFile ="~/MultipleContent.master"
    AutoEventWireup="true" CodeFile="MultipleContentPage.aspx.cs"
    Inherits="MultipleContentPage" Title="Content Page" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent"
runat="Server">
    This is the generic content for this page. Here you might provide
    some site
    specific text ... </asp:Content>

<asp:Content ID="Content2"
ContentPlaceHolderID="OtherLinksContent"
    runat="Server">
    Here's a<a href="http://www.prosetech.com">link</a>.<br />
    ...
</asp:Content>
```

9.8 MASTER PAGES AND RELATIVE PATHS:

- One quirk that can catch unsuspecting developers is the way that master pages handle relative paths.
- If all you're using is static text, this issue won't affect you.
- However, if you addError! Filename not specified.tags or any other HTML tag that points to another resource, problems can occur.
- The problem shows up if you place the master page in a different directory from the content page that uses it.
- This is a recommended best practice for large websites.
- In fact, Microsoft encourages you to use a dedicated folder for storing all your master pages.
- However, if you're not suitably careful, this can cause problems when you use relative paths.
- For example, imagine you put a master page in a subfolder named MasterPages and add the followingError! Filename not specified.tag to the master page:

- Assuming the file \MasterPages\banner.jpg exists, this appears to work fine. The image will even appear in the Visual Studio design environment.
- However, if you create a content page in another subfolder, the image path is interpreted relative to that folder.
- If the file doesn't exist there, you'll get a broken link instead of your graphic.
- Even worse, you could conceivably get the wrong graphic if another image has the same file name.
- To solve your problem, you could try to think ahead and write your URL relative to the content page where you want to use it.
- But this creates confusion and limits where your master page can be used.
- A better fix is to turn yourError! Filename not specified.tag into a server-side control, in which case ASP.NET will fix the mistake:

9.9 SUMMARY

- Building a professional web application involves much more than designing individual web pages. You also need the tools to integrate

your web pages in a complete, unified website. In this chapter, you considered best ways to do exactly that.

- ASP.NET themes feature, which lets you effortlessly apply a group of property settings to a control.
- Finally, you learned to use master pages, which allow you to standardize the layout of your website.
- All these features make it easy to bring your pages together into a well-integrated, consistent web application.

9.10 REFERANCE

- Beginning ASP.NET in C# by Matthew MacDonald

9.11 QUESTIONS

1. Explain themes and master pages in detail.
2. What is concept of master pages and content pages and how connect them?
3. Explain how themes works.
4. Explain how to handle theme conflicts.

❄❄❄❄❄❄❄

10

WEBSITE NAVIGATION

Unit Structure :

10.0 OBJECTIVES

- Website navigation is an essential part of web design because it contributes to the user experience.
- Understanding website navigation can help you allow users to access the information they want as quickly as possible by presenting an enjoyable, intuitive layout while increasing ease of use.

10.1 INTRODUCTION

- ASP.NET provides various site-navigation features which gives a consistent way for visitors to navigate the site.
- These features are Site Maps. URL Mapping and Routing, SiteMapPath.

10.2 SITE MAPS

- If your website has more than a handful of pages, you'll probably want some sort of navigation system to let users move from one page to the next.
- As with all the best ASP.NET features, ASP.NET navigation is flexible, configurable, and pluggable. It consists of three components:

    1. A way to define the navigation structure of your website. This part is the XML site map, which is (by default) stored in a file.

2. A convenient way to read the information in the site map file and convert it to an object model. The SiteMapDataSource control and the XmlSiteMapProvider perform this part.

3. A way to use the site map information to display the user's current position and give the user the ability to easily move from one place to another. This part takes place through the navigation controls you bind to the SiteMapDataSource control, which can include breadcrumb links, lists, menus, and trees.

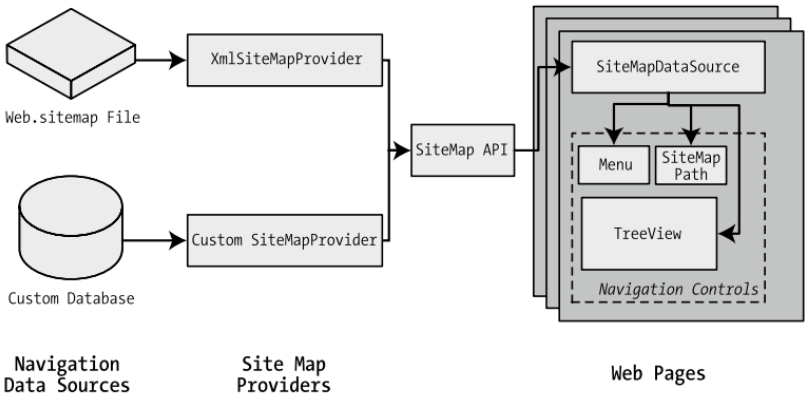


Figure 10.1: ASP.NET navigation with site maps
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- You can create a site map by using a text editor such as Notepad, or you can create it in Visual Studio by choosing Website ➤ Add New Item and then choosing the Site Map option.
- Either way, it's up to you to enter all the site map information by hand.
- The only difference is that if you create it in Visual Studio, the site map will start with a basic structure that consists of three siteMap nodes.

Rule 1: Site Maps Begin with the <siteMap> Element

- Every Web.sitemap file begins by declaring the element and ends by closing that element. You place the actual site map information between the start and end tags (where the three dots are shown here):

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  ...
</siteMap>
```

- The xmlns attribute is required, and must be entered exactly as shown here. This tells ASP.NET that the XML file uses the ASP.NET site map standard.

Rule 2: Each Page Is Represented by a <siteMapNode> Element

- Essentially, every site map defines an organization of web pages.
- To insert a page into the site map, you add the element with some basic information.
- Namely, you need to supply the title of the page, a description and the URL.
- You add these three pieces of information by using three attributes—named title, description, and url, as shown here:

```
<siteMapNode title="Home" description="Home" url="~/default.aspx" />
```

- Notice that this element ends with the characters />.
- This indicates it's an empty element that represents a start tag and an end tag in one.
- Empty elements never contain other nodes.
- Here's a complete, valid site map file that uses this page to define a website with exactly one page:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
    <siteMapNode title="Home" description="Home" url="~/default.aspx" />
</siteMap>
```

Rule 3: A <siteMapNode> Element Can Contain Other <siteMapNode> Elements

- Site maps don't consist of simple lists of pages. Instead, they divide pages into groups.
- To represent this in a site map file, you place one inside another.
- Instead of using the empty element syntax shown previously, you'll need to split your element into a start tag and an end tag:

```
<siteMapNode title="Home" description="Home" url="~/default.aspx">
 ...
</siteMapNode>
```

- Now you can slip more nodes inside. Here's an example of a Home group that contains two more pages:

```
<siteMapNode title="Home" description="Home" url="~/default.aspx">
        <siteMapNode title="Products" description="Our products"
```

```
        url="~/products.aspx" />
    <siteMapNode title="Hardware" description="Hardware choices"
        url="~/hardware.aspx" />
</siteMapNode>
```

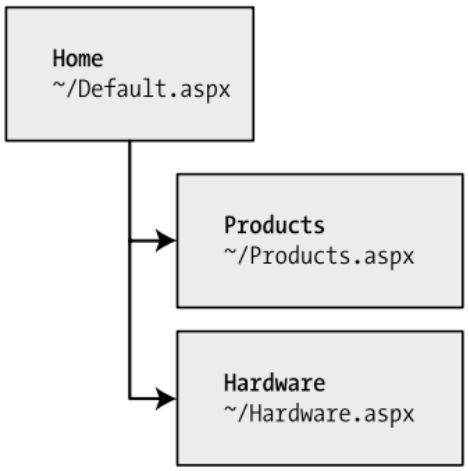


Figure 10.2: Three nodes in a site map
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- In this case, all three nodes are links. This means the user could surf to one of three pages.
- However, when you start to create more-complex groups and subgroups, you might want to create nodes that serve only to organize other nodes but aren't links themselves.
- In this case, just omit the url attribute, as shown here with the Products node:

```
<siteMapNode title="Products" description="Products">
    <siteMapNode title="In Stock" description="Products that are available"
        url="~/inStock.aspx" />
    <siteMapNode title="Not In Stock" description="Products that are on order"
        url="~/outOfStock.aspx" />
</siteMapNode>
```

Rule 4: Every Site Map Begins with a Single <siteMapNode>

- Another rule applies to all site maps.
- A site map must always have a single root node.

- All the other nodes must be contained inside this root-level node.
- That means the following is not a valid site map, because it contains two top-level nodes:

```
<siteMapNode title="Products" description="Our products"
 url="~/products.aspx" />
<siteMapNode title="Hardware" description="Hardware choices"
 url="~/hardware.aspx" />
```

- The following site map is valid, because it has a single top-level node (Home), which contains two more nodes:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
 <siteMapNode title="Home" description="Home" url="~/default.aspx">
    <siteMapNode title="Products" description="Our products"
      url="~/products.aspx" />
    <siteMapNode title="Hardware" description="Hardware choices"
      url="~/hardware.aspx" />
    </sitemapNode>
</siteMap>
```

Rule 5: Duplicate URLs Are Not Allowed

- You cannot create two site map nodes with the same URL.
- This might seem to present a bit of a problem when you want to have the same link in more than one place—and it does.
- However, it's a requirement because the default SiteMapProvider included with ASP.NET stores nodes in a collection, with each item indexed by its unique URL.
- This limitation doesn't prevent you from creating more than one URL with minor differences pointing to the same page. For example, consider the following portion of a site map.
- These two nodes are acceptable, even though they lead to the same page (products.aspx), because the two URLs have different query string arguments at the end.

```
<siteMapNode title="In Stock" description="Products that are available"
 url="~/products.aspx?stock=1" />
<siteMapNode title="Not In Stock" description="Products that are on order"
 url="~/products.aspx?stock=0" />
```

10.3 URL MAPPING AND ROUTING

URL Mapping

- In some situations, you might want to have several URLs lead to the same page.
- This might be the case for a number of reasons—maybe you want to implement your logic in one page and use query string arguments but still provide shorter and easier-to-remember URLs to your website users.
- The basic idea behind ASP.NET URL mapping is that you map a request URL to a different URL.
- The mapping rules are stored in the web.config file, and they're applied before any other processing takes place.
- Of course, for ASP.NET to apply the remapping, it must be processing the request, which means the request URL must use a file type extension that's mapped to ASP.NET.
- We can define URL mapping in the <urlMappings> section of the web.config file. You supply two pieces of information—the request URL (as the url attribute) and the new destination URL (mappedUrl). Here's an example:

```
<configuration>
 <system.web>
 <urlMappings enabled="true">
 <add url="~/category.aspx"
 mappedUrl="~/default.aspx?category=default" />
 <add url="~/software.aspx"
 mappedUrl="~/default.aspx?category=software" />
 </urlMappings>
 ...
 </system.web>
</configuration>
```

- In order for ASP.NET to make a match, the URL that the browser submits must match the URL specified in the web.config file almost exactly.
- When you use URL mapping, the redirection takes place in the same way as the Server.Transfer() method, which means no round-trip happens and the URL in the browser will still show the original request URL, not the new page.

URL ROUTING

- URL routing was originally designed as a core part of ASP.NET MVC, an alternative framework for building web pages.
- Unlike URL mapping, URL routing doesn't take place in the web.config file. Instead, it's implemented using code.
- Typically, you'll use the Application_Start() method in the global.asax file to register all the routes for your application.
- To register a route, you use the RouteTable class from the System.Web.Routing namespace. To make life easier, you can start by importing that namespace:
  using System.Web.Routing;
- The RouteTable class provides a static property named Routes, which holds a collection of Route objects that are defined for your application.
- Initially, this collection is empty, but you can create custom routes by calling the MapPageRoute() method, which takes three arguments:
  - routeName: This is a name that uniquely identifies the route. It can be whatever you want.
  - routeUrl: This specifies the URL format that browsers will use. Typically, a route URL consists of one or more pieces of variable information, separated by slashes, which are extracted and provided to your code. For example, you might request a product page by using a URL such as /products/4312.
  - physicalFile: This is the target web form—the place where users will be redirected when they use the route. The information from the original routeUrl will be parsed and made available to this page as a collection through the Page.RouteData property.
- Here's an example that adds two routes to a web application when it first starts:

```
protected void Application_Start(object sender, EventArgs e)
{
 RouteTable.Routes.MapPageRoute("product-details",
 "product/{productID}", "~/productInfo.aspx");
 RouteTable.Routes.MapPageRoute("products-in-category",
 "products/category/{categoryID}", "~/products.aspx");
}
```

10.4 SITEMAPPATH CONTROL

- The TreeView shows the available pages, but it doesn't indicate where you're currently positioned. To solve this problem, it's common to use the TreeView in conjunction with the SiteMapPath control.

  <asp:SiteMapPath ID="SiteMapPath1" runat="server" />

- The SiteMapPath provides breadcrumb navigation, which means it shows the user's current location and allows the user to navigate up the hierarchy to a higher level by using links.

- Using the SiteMapPath control, the user can return to the default.aspx page.

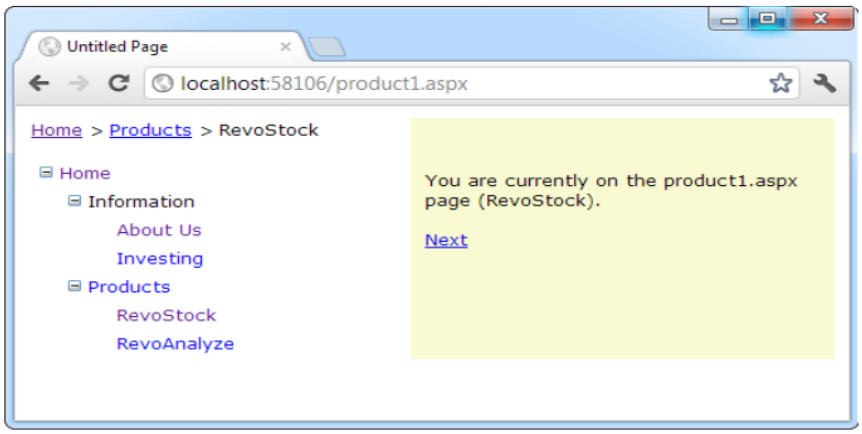


Figure 10.3: Breadcrumb navigation with SiteMapPath
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Lists some of its most commonly configured properties.

Property	Description
ShowToolTips	Set this to false if you don't want the description text to appear when the user hovers over a part of the site map path.
ParentLevelsDisplayed	This sets the maximum number of levels above the current page that will be shown at once. By default, this setting is –1, which means all levels will be shown.
RenderCurrentNodeAsLink	If true, the portion of the page that indicates the current page is turned into a clickable link. By default, this is false because the user is already at the current page.

PathDirection	You have two choices: RootToCurrent (the default) and CurrentToRoot (which reverses the order of levels in the path).
PathSeparator	This indicates the characters that will be placed between each level in the path. The default is the greater-than symbol (>). Another common path separator is the colon (:).

Table 10.1: SiteMapPath Appearance-Related Properties
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

Using SiteMapPath Styles and Templates

Style	Template	Applies To
NodeStyle	NodeTemplate	All parts of the path except the root and current node.
CurrentNodeStyle	CurrentNodeTemplate	The node representing the current page.
RootNodeStyle	RootNodeTemplate	The node representing the root. If the root node is the same as the current node, the current node template or styles are used.
PathSeparatorStyle	PathSeparatorTemplate	The separator between each node

Table 10.2: SiteMapPath Styles and Templates
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

10.5 TREEVIEW CONTROL

- The TreeView has a slew of properties that let you change how it's displayed on the page. One of the most important properties is ImageSet, which lets you choose a predefined set of node icons.
- The TreeView offers 16 possible ImageSet values, which are represented by the TreeViewImageSet enumeration.
- For example, following Figure shows the same RevoStock navigation page you considered earlier, but this time with an ImageSet value of TreeViewImageSet.Faq.

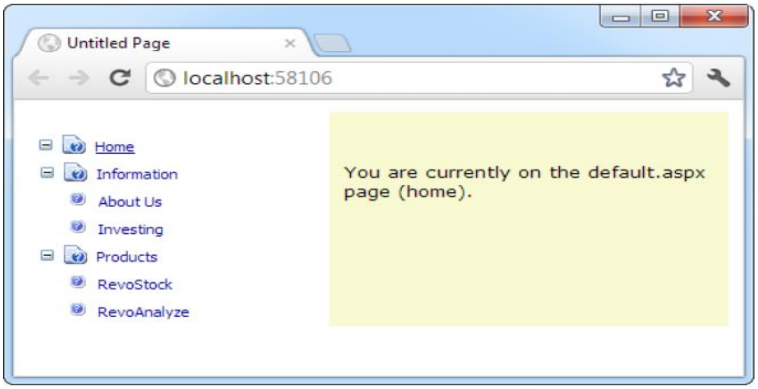


Figure 10.4: A TreeView with fancy node icons
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Here's the complete TreeView markup:

```
<asp:TreeView ID="TreeView1" runat="server"
 DataSourceID="SiteMapDataSource1"  ImageSet="Faq"
NodeIndent="0" >
</asp:TreeView>
```

10.5.1 Useful TreeView Properties

Property	Description
MaxDataBindDepth	Determines how many levels the TreeView will show. By default, MaxDataBindDepth is –1, and you'll see the entire tree.
ExpandDepth	Lets you specify how many levels of nodes will be visible at first. If you use 0, the TreeView begins completely closed.
NodeIndent	Sets the number of pixels between each level of nodes in the TreeView. Set this to 0 to create a nonindented TreeView, which saves space.
ImageSet	Lets you use a predefined collection of node images for collapsed, expanded, and nonexpandable nodes.
ShowLines	Adds lines that connect every node when set to true.
NodeWrap	Lets a node text-wrap over more than one line when set to true.
ShowCheckBoxes	Shows a check box next to every node when set to true. This isn't terribly useful for site maps, but it is useful with other types of trees.

Table 10.3: Useful TreeView Properties
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

10.5.2 TreeView Styles

- Styles are represented by the TreeNodeStyle class, which derives from the more conventional Style class.
- As with other rich controls, the styles give you options to set background and foreground colors, fonts, and borders.

TreeNodeStyle-Added Properties

Property	Description
ImageUrl	The URL for the image shown next to the node.
NodeSpacing	The space (in pixels) between the current node and the node above and below.
VerticalPadding	The space (in pixels) between the top and bottom of the node text and border around the text.
HorizontalPadding	The space (in pixels) between the left and right of the node text and border around the text.
ChildNodesPadding	The space (in pixels) between the last child node of an expanded parent node and the following node.

Table 10.4: TreeNodeStyle-Added Properties
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

10.6 MENU CONTROL

- The Menu control is another rich control that supports hierarchical data. Like the TreeView, you can bind the Menu control to a data source, or you can use MenuItem objects to fill it by hand.
  <asp:Menu ID="Menu1" runat="server"
  DataSourceID="SiteMapDataSource1" />

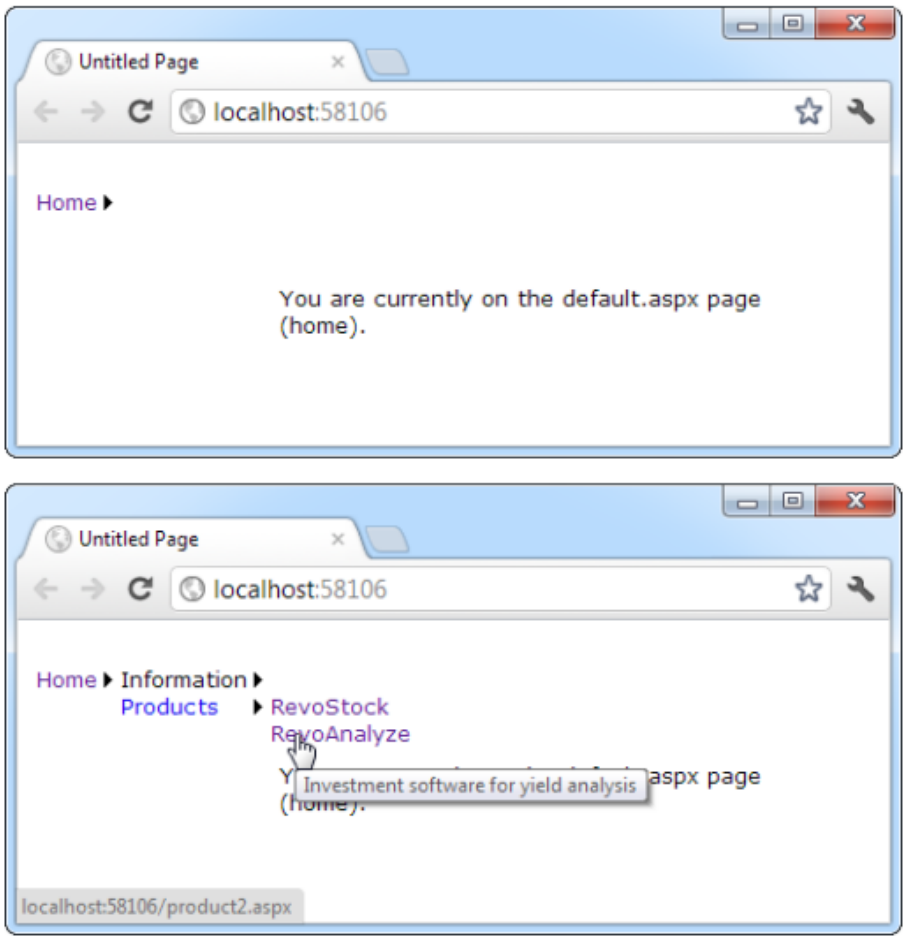


Figure 10.5: Navigating through the menu
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

- Overall, the Menu and TreeView controls expose strikingly similar programming models, even though they render themselves quite differently.
- They also have a similar style-based formatting model. But a few noteworthy differences exist:
  - The Menu displays a single submenu. The TreeView can expand an arbitrary number of node branches at a time.
  - The Menu displays a root level of links in the page. All other items are displayed using fly-out menus that appear over any other content on the page. The TreeView shows all its items inline in the page.
  - The Menu supports templates. The TreeView does not. (Menu templates are discussed later in this section.)
  - The TreeView supports check boxes for any node. The Menu does not.

- The Menu supports horizontal and vertical layouts, depending on the Orientation property. The TreeView supports only vertical layout.

Menu Styles

Static Style	Dynamic Style	Description
StaticMenuStyle	DynamicMenuStyle	Sets the appearance of the overall "box" in which all the menu items appear. In the case of StaticMenuStyle, this box appears on the page, and with DynamicMenuStyle, it appears as a pop-up.
StaticMenuItemStyle	DynamicMenuItemStyle	Sets the appearance of individual menu items.
StaticSelectedStyle	DynamicSelectedStyle	Sets the appearance of the selected item. Note that the selected item isn't the item that's currently being hovered over; it's the item that was previously clicked
StaticHoverStyle	DynamicHoverStyle	Sets the appearance of the item that the user is hovering over with the mouse.

Table 10.5: Menu Styles
(Ref: Beginning ASP.NET in C# by Matthew MacDonald)

10.7 SUMMARY

- In this chapter, you explored the new navigation model and learned how to define site maps and bind the navigation data.
- You then considered three controls that are specifically designed for navigation data: the SiteMapPath, TreeView, and Menu.
- Using these controls, you can add remarkably rich site maps to your websites with very little coding.

- But before you begin, make sure you've finalized the structure of your website.
- Only then will you be able to create the perfect site map and choose the best ways to present the site map information in the navigation controls.

10.8 REFERANCE

- Beginning ASP.NET in C# by Matthew MacDonald

10.9 QUESTIONS

1. Explain SiteMapPath control in detail.
2. Write note on menu control.
3. Write note on TreeView control.
4. Write and explain Menu style.
5. Write and explain TreeView properties.

❄❄❄❄❄❄❄

11

ADO.NET

Unit Structure :

11.0 OBJECTIVE

1. To understand the basic and advanced concepts of ADO.NET.
2. To learn how to establish connection between application and data sources.
3. To study the different components of ADO.NET that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.
4. To learn how to prevent SQL injection attacks and how to use transactions.

11.1 INTRODUCTION

- ADO.NET consists of managed classes that allow .NET applications to connect to data sources (relational databases), execute commands, and manage disconnected data.
- The small miracle of ADO.NET is that it enables you to write more or less the same data access code in web applications that you write for client-server desktop applications, or even single-user applications that connect to a local database.
- This shows the implementation of the architecture of ADO.NET and the ADO.NET data providers.
- ADO.NET describes such as opening a connection, executing a SQL statement or stored procedure, and retrieving the results of a query.

11.2 DATA PROVIDER MODEL

- ADO.NET uses a multilayered architecture that revolves around a few key concepts, such as Connection, Command, and DataSet objects.
- One of the key differences between ADO.NET and some other database technologies is how it deals with the challenge of different data sources.
- In many past database technologies, such as classic ADO, programmers use a generic set of objects no matter what the underlying data source is. For example, if you want to retrieve a record from an Oracle database using ADO code, we use the same Connection class you would use to tackle the task with SQL Server.
- ADO.NET Data Providers A data provider is a set of ADO.NET classes that allows you to access a specific database, execute SQL commands, and retrieve data.
- A data provider is a bridge between your application and a data source. The classes that make up a data provider include the following:
    - Connection: You use this object to establish a connection to a data source.
    - Command: You use this object to execute SQL commands and stored procedures.
    - DataReader: This object provides fast read-only, forward-only access to the data retrieved from a query.
    - DataAdapter: This object performs two tasks. First, you can use it to fill a DataSet with information extracted from a data source. Second, you can use it to apply changes to a data source, according to the modifications made in a DataSet.
- ADO.NET doesn't include generic data provider objects. Instead, it includes different data providers specifically designed for different types of data sources. Each data provider has a specific implementation of the Connection, Command, DataReader, and DataAdapter classes that's optimized for a specific RDBMS. For example, if you need to create a connection to a SQL Server database, use a connection class named SqlConnection.
- Developers can create their own providers for proprietary data sources. In fact, numerous proof-of-concept examples are available that show how you can easily create custom ADO.NET providers to wrap non relational data stores, such as the file system or a directory service. Some third-party vendors also sell custom providers for .NET

- The .NET Framework is bundled with a small set of four providers:
  - SQL Server provider: Provides optimized access to a SQL Server database
  - OLE DB provider: Provides access to any data source that has an OLE DB driver.
  - Oracle provider: Provides optimized access to an Oracle database
  - ODBC provider: Provides access to any data source that has an ODBC driver.

Figure 1 shows the layers of the ADO.NET provider model.

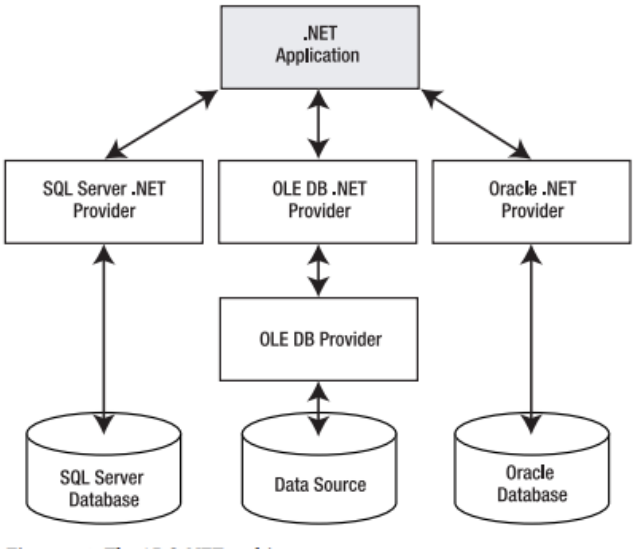


Figure 1 The ADO.NET architecture

11.3 DIRECT DATA ACCESS

- ADO.NET does not provide a single set of objects that communicate with multiple database management systems (DBMSs).
- ADO.NET supports multiple data providers, each of which is optimized to interact with a specific DBMS.
- The first benefit of this approach is that you can program a specific data provider to access any unique features of a particular DBMS.
- The second benefit is that a specific data provider can connect directly to the underlying engine of the DBMS in question without an intermediate mapping layer standing between the tiers to communicate with a specific type of data source.
- Following Table 1 shows documents of some of the core common types, their base class and the key interfaces they implement.

Type of Object	Base Class	Relevant Interfaces	Meaning in Life
Connection	DbConnection	IDbConnection	Provides the ability to connect to and disconnect from the data store. Connection objects also provide access to a related transaction object.
Command	DbCommand	IDbCommand	Represents a SQL query or a stored procedure. Command objects also provide access to the provider's data reader object.
DataReader	DbDataReader	IDataReader, IDataRecord	Provides forward-only, read-only access to data using a server-side cursor.
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Transfers DataSets between the caller and the data store. Data adapters contain a connection and a set of four internal command objects used to select, insert, update, and delete information from the data store.
Parameter	DbParameter	IDataParameter, IDbDataParameter	Represents a named parameter within a parameterized query.
Transaction	DbTransaction	IDbTransaction	Encapsulates a database transaction

Table 1 Core Common Types

- Figure 2 shows the ADO.NET data providers.

- A data provider will supply with other types beyond the objects shown in Figure 2 however, these core objects define a common baseline across all data providers.

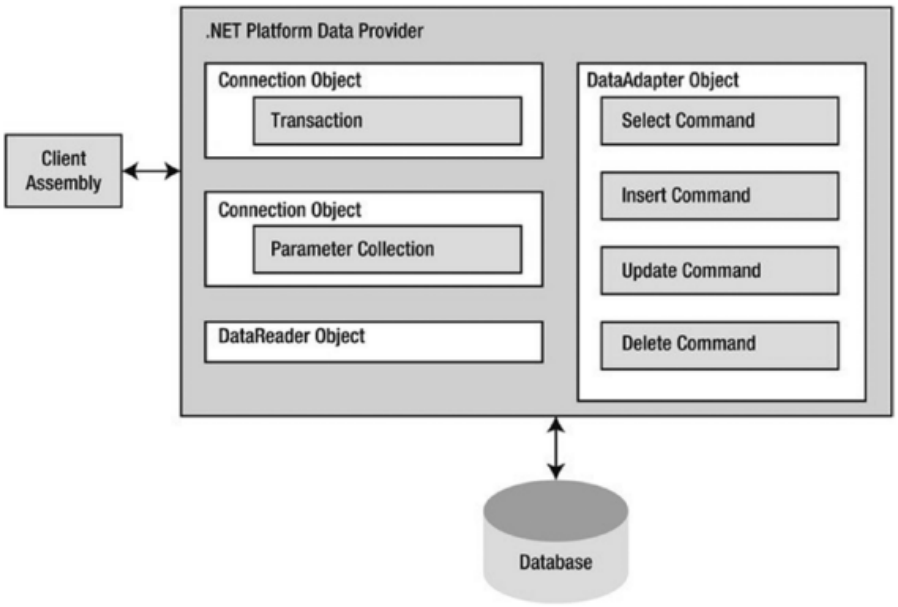


Figure 2. ADO.NET data providers provide access to a given DBMS

ADO.NET Data Providers

- A .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results.
- Those results are either processed directly, placed in a DataSet in order to be exposed to the user as needed, combined with data from multiple sources, or remoted between tiers. .NET Framework data providers are lightweight, creating a minimal layer between the data source and code, increasing performance without sacrificing functionality.

The following table 2 lists the data providers that are included in the .NET Framework.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server. Uses the System.Data.SqlClient namespace.
.NET Framework Data Provider for OLE DB	For data sources exposed by using OLE DB. Uses the System.Data.OleDb namespace.

.NET Framework data provider	Description
.NET Framework Data Provider for ODBC	For data sources exposed by using ODBC. Uses the System.Data.Odbc namespace.
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the System.Data.OracleClient namespace.
EntityClient Provider	Provides data access for Entity Data Model (EDM) applications. Uses the System.Data.EntityClient namespace.
.NET Framework Data Provider for SQL Server Compact 4.0.	Provides data access for Microsoft SQL Server Compact 4.0. Uses the System.Data.SqlServerCe namespace.

Table 2 lists of data providers

Core Objects of .NET Framework Data Providers

The following table 3 outlines the four core objects that make up a .NET Framework data provider.

Object	Description
Connection	Establishes a connection to a specific data source. The base class for all Connection objects is the DbConnection class.
Command	Executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection. The base class for all Command objects is the DbCommand class.
DataReader	Reads a forward-only, read-only stream of data from a data source. The base class for all DataReader objects is the DbDataReader class.
DataAdapter	Populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class.

Table 3 Core objects

A .NET Framework data provider also contains the classes listed in the following table 4.

Object	Description
Transaction	Enlists commands in transactions at the data source. The base class for all Transaction objects is the DbTransaction class. ADO.NET also provides support for transactions using classes in the System.Transactions namespace.
CommandBuilder	A helper object that automatically generates command properties of a DataAdapter or derives parameter information from a stored procedure and populates the Parameters collection of a Command object. The base class for all CommandBuilder objects is the DbCommandBuilder class.
ConnectionStringBuilder	A helper object that provides a simple way to create and manage the contents of connection strings used by the Connection objects. The base class for all ConnectionStringBuilder objects is the DbConnectionStringBuilder class.
Parameter	Defines input, output, and return value parameters for commands and stored procedures. The base class for all Parameter objects is the DbParameter class.
Exception	Returned when an error is encountered at the data source. For an error encountered at the client, .NET Framework data providers throw a .NET Framework exception. The base class for all Exception objects is the DbException class.
Error	Exposes the information from a warning or error returned by a data source.
ClientPermission	Provided for .NET Framework data provider code access security attributes. The base class for all ClientPermission objects is the DBDataPermission class.

Table 4 Data Providers Classes

NET Framework Data Provider for SQL Server (SqlClient)

- The .NET Framework Data Provider for SQL Server (SqlClient) uses its own protocol to communicate with SQL Server.

- It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or Open Database Connectivity (ODBC) layer.
- The following figure 3 shows the .NET Framework Data Provider for SQL Server with the .NET Framework Data Provider for OLE DB.
- The .NET Framework Data Provider for OLE DB communicates to an OLE DB data source through both the OLE DB Service component, which provides connection pooling and transaction services, and the OLE DB provider for the data source.

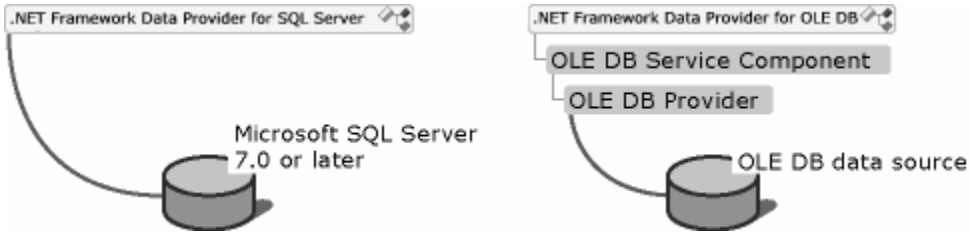


Figure 3 Comparison of the .NET Framework Data Provider for SQL Server and the .NET Framework Data Provider for OLE DB

- The .NET Framework Data Provider for SQL Server classes are located in the System.Data.SqlClient namespace.
- The .NET Framework Data Provider for SQL Server supports both local and distributed transactions. For distributed transactions, the .NET Framework Data Provider for SQL Server, by default, automatically enlists in a transaction and obtains transaction details from Windows Component Services or System.Transactions. For more information, see Transactions and Concurrency.

The following code example 1 shows how to include the System.Data.SqlClient namespace in your applications.

```
using System.Data.SqlClient;
```

Example 1 System.Data.SqlClient

.NET Framework Data Provider for OLE DB

- The .NET Framework Data Provider for OLE DB (OleDb) uses native OLE DB through COM interop to enable data access.
- The .NET Framework Data Provider for OLE DB supports both local and distributed transactions.
- For distributed transactions, the .NET Framework Data Provider for OLE DB, by default, automatically enlists in a transaction and obtains transaction details from Windows Component Services. For more information, see Transactions and Concurrency.

The following Table 5 shows the providers that have been tested with ADO.NET.

Driver	Provider
SQLOLEDB	Microsoft OLE DB provider for SQL Server
MSDAORA	Microsoft OLE DB provider for Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB provider for Microsoft Jet

Table 5 Data Provider for OLE DB

.NET Framework Data Provider for ODBC

- The .NET Framework Data Provider for ODBC (Odbc) uses the native ODBC Driver Manager (DM) to enable data access.
- The ODBC data provider supports both local and distributed transactions. For distributed transactions, the ODBC data provider, by default, automatically enlists in a transaction and obtains transaction details from Windows Component Services.

The following Table 6 shows the ODBC drivers tested with ADO.NET.

Driver
SQL Server
Microsoft ODBC for Oracle
Microsoft Access Driver (*.mdb)

Table 6 ODBC Drivbers

.NET Framework Data Provider for Oracle

- The .NET Framework Data Provider for Oracle enables data access to Oracle data sources through Oracle client connectivity software.
- The data provider supports Oracle client software version 8.1.7 or a later version. The data provider supports both local and distributed transactions.
- The .NET Framework Data Provider for Oracle requires Oracle client software (version 8.1.7 or a later version) on the system before you can connect to an Oracle data source.
- .NET Framework Data Providers for Oracle classes are located in the System.Data.OracleClient namespace and are contained in the System.Data.OracleClient.dll assembly. You must reference both the System.Data.dll and the System.Data.OracleClient.dll when you compile an application that uses the data provider.

The following code example 2 shows how to include the System.Data.OracleClient namespace in your applications.

Example 2 System.Data.OracleClient

```
using System.Data;
using System.Data.OracleClient;
```

11.4 CREATING A CONNECTION

- To connect to Microsoft SQL Server, use the SqlConnection object of the .NET Framework Data Provider for SQL Server.
- To connect to an OLE DB data source, use the OleDbConnection object of the .NET Framework Data Provider for OLE DB.
- To connect to an ODBC data source, use the OdbcConnection object of the .NET Framework Data Provider for ODBC.
- To connect to an Oracle data source, use the OracleConnection object of the .NET Framework Data Provider for Oracle.

Choosing a .NET Framework Data Provider

- Depending on the design and data source for your application, .NET Framework data provider can improve the performance, capability, and integrity of your application.
- The following Table 7 discusses the advantages and limitations of each .NET Framework data provider.

Table 7 .NET Framework data provider

Provider	Notes
.NET Framework Data Provider for SQL Server	Recommended for middle-tier applications that use Microsoft SQL Server. Recommended for single-tier applications that use Microsoft Database Engine (MSDE) or SQL Server. Recommended over use of the OLE DB provider for SQL Server (SQLOLEDB) with the .NET Framework Data Provider for OLE DB.
.NET Framework Data Provider for OLE DB	For SQL Server, the .NET Framework Data Provider for SQL Server is recommended instead of this provider. Recommended for single-tier applications that use Microsoft Access databases. Use of an Access database for a middle-tier application is not recommended.
.NET Framework	Recommended for middle and single-tier applications

Data Provider for ODBC	that use ODBC data sources.
.NET Framework Data Provider for Oracle	Recommended for middle and single-tier applications that use Oracle data sources.

11.5 SELECT COMMAND

The command object is one of the basic components of ADO .NET.

1. The Command Object uses the connection object to execute SQL queries.
2. The queries can be in the Form of Inline text, Stored Procedures or direct Table access.
3. An important feature of Command object is that it can be used to execute queries and Stored Procedures with Parameters.
4. If a select query is issued, the result set it returns is usually stored in either a DataSet or a DataReader object.

The properties associated with the SqlCommand class are shown in the Table 8 below.

Property	Type of Access	Description
Connection	Read/Write	The SqlConnection object that is used by the command object to execute SQL queries or Stored Procedure.
CommandText	Read/Write	Represents the T-SQL Statement or the name of the Stored Procedure.
CommandType	Read/Write	This property indicates how the CommandText property should be interpreted. The possible values are: 1. Text (T-SQL Statement) 2. StoredProcedure (Stored Procedure Name) 3. TableDirect
CommandTimeout	Read/Write	This property indicates the time to wait when executing a particular command. Default Time for Execution of Command is 30 Seconds. The Command is aborted after it times out and an exception is thrown.

Table 8 Properties of SqlCommand class

- Various Execute Methods that can be called from a Command Object. Shown in the following Table 9.

Property	Description
ExecuteNonQuery	This method executes the command specified and returns the number of rows affected.
ExecuteReader	The ExecuteReader method executes the command specified and returns an instance of SqlDataReader class.
ExecuteScalar	This method executes the command specified and returns the first column of the first row of the result set. The remaining rows and columns are ignored.
ExecuteXMLReader	This method executes the command specified and returns an instance of XmlReader class. This method can be used to return the result set in the form of an XML document

Table 9 Properties of execute methods

ExecuteNonQuery

1. The ExecuteNonQuery method is used to execute the command and return the number of rows affected.
2. The ExecuteNonQuery method cannot be used to return the result set. (Shown in example 3)

Example 3

```
public void CallExecuteNonQuery()
{
   SqlConnection conn = new SqlConnection();
   conn.ConnectionString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;
   try
   {
      SqlCommand cmd = new SqlCommand();
      cmd.Connection = conn;
      cmd.CommandText = "DELETE FROM EMP WHERE DEPTNO =
```

```
40";
        cmd.CommandType = CommandType.Text;
        conn.Open();
        Int32 RowsAffected = cmd.ExecuteNonQuery();
        MessageBox.Show(RowsAffected + " rows affected", "Message");
        cmd.Dispose();
        conn.Dispose();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

11.6 DATAREADER

- A data reader provides an easy way for the programmer to read data from a database as if it were coming from a stream.
- The DataReader is the solution for forward streaming data through ADO.NET.
- The data reader is also called a firehose cursor or forward read-only cursor because it moves forward through the data.
- The data reader not only allows you to move forward through each record of the database, but it also enables you to parse the data from each column.
- The DataReader class represents a data reader in ADO.NET.
- Similar to other ADO.NET objects, each data provider has a data reader class for example; OleDbDataReader is the data reader class for OleDb data providers. Similarly, SqlDataReader and ODBC DataReader are data reader classes for SQL and ODBC data providers, respectively.
- The IDataReader interface defines the function of a data reader and works as the base class for all data provider-specific data reader classes such as OleDataReader. SqlDataReader, and OdbcDataReader. Figure 4 shows some of the classes that implement IDbDataReader.

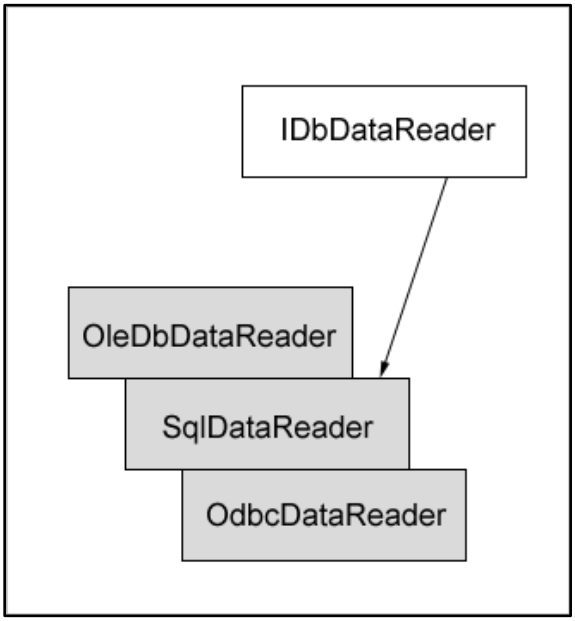


Figure 4. Data Provider-specific classes implementing IdbDataReader

Initializing DataReader

- call the ExecuteReader method of the Command object, which returns an instance of the DataReader.
- For example 4, use the following line of code:

Example 4:

```
SqlCommand cmd = new SqlCommand(SQL, conn);
// Call ExecuteReader to return a DataReader
SqlDataReader reader = cmd.ExecuteReader();
```

Once you're done with the data reader, call the Close method to close a data reader:

```
reader.Close();
```

DataReader Properties(Shown in Table 10)

Property	Description
Depth	Indicates the depth of nesting for row
FieldCount	Returns number of columns in a row
IsClosed	Indicates whether a data reader is closed
Item	Gets the value of a column in native format
RecordsAffected	Number of row affected after a transaction

Table 10 DataReader Properties

The DataReader methods (Shown in Table 11)

Method	Description
Close	Closes a DataRaeder object.
Read	Reads next record in the data reader.
NextResult	Advances the data reader to the next result during batch transactions.
Getxxx	There are dozens of Getxxx methods. These methods read a specific data type value from a column. For example. GetChar will return a column value as a character and GetString as a string.

Table 11 DataReader *methods*

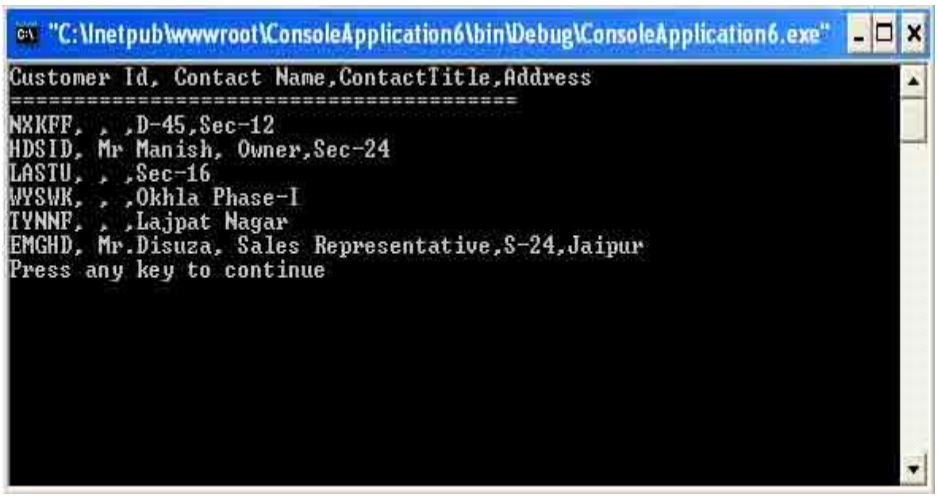
Reading with the DataReader

- The OleDbDataReader is initialized, utilizing its various methods to read data records.
- Read method, which, when called repeatedly, continues to read each row of data into the DataReader object.
- The DataReader also provides a simple indexer that enables to pull each column of data from the row.

Example 5: create a connection object, create a command object, called the ExecuteReader method, called the DataReader's Read method until the end of the data, and then display the data. At the last, released the data reader and connection objects.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;

namespace CommandTypeEnumeration
{
   class Program
   {
      static void Main(string[] args)
      {
         // Create a connection string
         string ConnectionString = "Integrated Security = SSPI; " +
         "Initial Catalog= Northwind; " + " Data source = localhost; ";
         string SQL = "SELECT * FROM Customers";
```

```
        // create a connection object
        SqlConnection conn = new SqlConnection(ConnectionString);

        // Create a command object
        SqlCommand cmd = new SqlCommand(SQL, conn);
        conn.Open();

        // Call ExecuteReader to return a DataReader
        SqlDataReader reader = cmd.ExecuteReader();
        Console.WriteLine("customer ID, Contact Name, " + "Contact Title, Address ");
        Console.WriteLine("============================");
          while (reader.Read())
        {
           Console.Write(reader["CustomerID"].ToString() + ", ");
           Console.Write(reader["ContactName"].ToString() + ", ");
           Console.Write(reader["ContactTitle"].ToString() + ", ");
           Console.WriteLine(reader["Address"].ToString() + ", ");
        }

        //Release resources
        reader.Close();
        conn.Close();
      }
   }
}
```



11.7 DISCONNECTED DATA ACCESS

- The ADO.NET Framework supports two models of Data Access Architecture, Connection Oriented Data Access Architecture and Disconnected Data Access Architecture.
- The ADO.NET Disconnected Data Access Architecture is far more flexible and powerful than ADOs Connection Oriented Data Access.
- In Connection Oriented Data Access Architecture the application makes a connection to the Data Source and then interacts with it through SQL requests using the same connection.
- The application stays connected to the database system even when it is not using any Database Operations.
- The DataSet is the central component in the ADO.NET Disconnected Data Access Architecture.
- A DataSet is an in-memory data store that can hold multiple tables at the same time.

```
DataSet ds = new DataSet();
```

11.8 SUMMARY

1. ADO.NET consists of managed classes that allow .NET applications to connect to data sources (relational databases), execute commands, and manage disconnected data.
2. ADO.NET uses a multilayered architecture that revolves around a few key concepts, such as Connection, Command, and DataSet objects.
3. The .NET Framework Data Provider for SQL Server (SqlClient) uses its own protocol to communicate with SQL Server.
4. A data reader provides an easy way for the programmer to read data from a database as if it were coming from a stream.
5. The ADO.NET Framework supports two models of Data Access Architecture, Connection Oriented Data Access Architecture and Disconnected Data Access Architecture.

11.9 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

11.10 UNIT END EXERCISES

1. What is ADO.NET? Explain the Data Provider Model?
2. Explain the ADO.NET Direct Data Access, with an example?
3. Write a short note on:

a. Creating a Connection

b. Select Command

c. DataReader

d. Disconnected Data Access

❄❄❄❄❄❄❄

12

DATA BINDING

Unit Structure :

12.0 OBJECTIVE

1. To learn how data binding and the data source controls work.
2. To understand the single and repeated value data binding.
3. To study the SqlDataSource controls.

12.1 INTRODUCTION

- Data binding is an aspect that allows us to associate a data source with a control and have that control automatically display data.
- The main characteristic of data binding is that it's declarative, not programmatic. That means data binding is defined outside program, alongside the controls in the .aspx page.
- The benefit is that it helps us achieve a cleaner separation between controls and program in a web page.
- In ASP.NET, most web controls (including TextBox, LinkButton, Image, and many more) support single-value data binding. With single-value binding, we can bind a control property to a data source, but the control can display only a single value.
- The property we bind doesn't need to represent something directly visible on the page.
- For example, not only can we bind the text of a hyperlink by setting the Hyperlink.Text property, but we can also bind the NavigateUrl property to specify the target destination of the link. To use single-value binding, we create data binding expressions.

- Numerous web controls support repeated value binding, which means they can provide a set of items.
- Repeated value controls often create lists and grids (e.g. the ListBox and GridView are two examples).
- If a control supports repeated-value binding, it always uncovers a DataSource property, which accepts a data object.
- After setting the DataSource property, create the logical link from the server control to the data object that contains the data to render.
- The control's DataBind() method, which loops through the data source, extracts its data, and provides it to the page. Repeated-value binding is by far the more poful type of data binding.

12.2 SINGLE-VALUE DATA BINDING

- The controls that support single-value data binding allow us to bind some of their properties to a data binding expression.
- This expression is entered in the .aspx markup portion of the page (not the program- behind file) and enclosed between the <%# and %> delimiters. Here's an syntax:
  <%# expression_goes_here %>
- This looks like a script block, but it isn't. While coding inside this tag, error will get The only thing we can add is valid data binding expressions. For example, if we have a public, protected, or internal variable in page class named EmployeeName, we could write the following:
  <%# EmployeeName %>
- To evaluate a data binding expression, you must call the Page.DataBind() method in program.
- While calling DataBind(), ASP.NET will examine all the expressions on page and replace them with the corresponding value.
- If we forget to call the DataBind() method, the data binding expression won't be filled in instead, it just gets thrown away when page is rendered to HTML.
- The source for single-value data binding can include the value of a
  - property,
  - member variable,
  - return value of a function

  (as long as the property, member variable, or function has an accessibility of protected, public, or internal).

- It can also be any other expression that can be evaluated at runtime, such as a reference to another control's property, a calculation using operators and literal values, and so on.
- The valid data binding expressions:

  <%# GetUserName() %>

  <%# 3 + (3 * 40) %>

  <%# "amit " + "raju" %>

  <%# Request.Browser.Browser %>

- Add data binding expressions just about anywhere on the page, then assign a data binding expression to a property in the control tag.
- Example 1 page that uses several data binding expressions:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<form method="post" runat="server">
<asp:Image ID="image1" runat="server" ImageUrl='<%# FilePath %>' />
<br />
<asp:Label ID="label1" runat="server" Text='<%# FilePath %>' />
<br />
<asp:TextBox ID="textBox1" runat="server" Text='<%# GetFilePath() %>' />
<br />
<asp:HyperLink ID="hyperLink1" runat="server"
NavigateUrl='<%# LogoPath.Value %>' Font-Bold="True" Text="Show logo" />
<br />
<input type="hidden" ID="LogoPath" runat="server" value="apress.gif">
<b><%# FilePath %></b><br />
<img src="<%# GetFilePath() %>">
</form>
</body>
</html>
```

Example 1 Data binding expressions

- We bind the Text property of a Label and a TextBox & use other properties such as the ImageUrl of an Image, the NavigateUrl property of a HyperLink, and the src attribute of a static HTML <img> tag.
- After we put the binding expression elsewhere in the page without binding to any property or attribute. For example, the previous web page has a binding expression between the and tags.

- While it's executing, the resulting text will be rendered on the page and rendered in bold type. Then place the expression outside the <form> section, as long as you don't try to insert a server-side control on a page.

- The expressions in this sample page refer to a FilePath property, a GetFilePath() function, and the Value property of a server-side hidden field that's declared on the same page. To complete this page, need to define these ingredients in script blocks or in the program-behind class: (Shown in Example 2)

```
protected string GetFilePath()
{
return "rose.gif";
}
protected string FilePath
{
get { return "rose.gif"; }
}
```

Example 2 FilePath property

In this program, the property and function return only a hard-program string. However, we can also add just about any C# program to generate the value for the data binding expression dynamically.

12.3 REPEATED-VALUE DATA BINDING,

- Repeated-value binding allows to bind an entire list of information to a control. This list of information is represented by a data object that wraps a collection of items. This could be a collection of convention objects (e.g., an ordinary ArrayList or Hashtable) or a collection of rows (for e.g., with a DataReader or DataSet). ASP.NET includes several basic list controls that support repeated-value binding:
    - All controls that render themselves using the <select> tag, including the HtmlSelect, ListBox, and DropDownList controls
    - The CheckBoxList and RadioButtonList controls, which render each child item with a separate checkbox or radio button.
    - The BulletedList control, which creates a list of bulleted or numbered points
- All controls display a single-value field of a property from each data item. When performing data binding with one of these controls, use the following listed properties.

Data Properties for List Controls

1. DataSource

This is a data object that contains a collection of data items to display. This data object must implement one of the interfaces that ASP.NET data binding supports, typically ICollection.

2. **DataSourceID**

Instead of supplying the data object programmatically, link list control to a data source control by setting this property. The data source control will generate the required data object automatically. Use DataSource property or the DataSourceID property, but not both.

3. DataTextField

Every data source represents a collection of data items. A list control displays only a single Value from each list item. The DataTextField denotes the field (row) or property (object) of the data item that contains the value to display in the page.

4. DataTextFormatString

This property states an optional format string that the control will use to format each DataTextValue before displaying it. For example, state that a number should be formatted as a currency value.

5. DataValueField

This property is related to the DataTextField property, but the value from the data item isn't displayed in the page; rather, it's stored in the value attribute of the underlying HTML tag. This enables retrieving the value afterwards in program. The primary use of this field is to store a unique ID or primary key field can use it later to retrieve more data when the user selects a specific item.

Program for creating and binding the hashtable(shown in example 3):

Example 3

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (!Page.IsPostBack)
{
// Create the data source.
Hashtable ht = new Hashtable();
ht.Add("Key1", "red");
ht.Add("Key2", "blue");
ht.Add("Key3", "Pink");
```

```
// Set the DataSource property for the controls.
Select1.DataSource = ht;
Select2.DataSource = ht;
Listbox1.DataSource = ht;
DropdownList1.DataSource = ht;
CheckList1.DataSource = ht;
OptionList1.DataSource = ht;
// Bind the controls.
this.DataBind();
}
```

Binding to a DataReader

- The above program used a hashtable as the data source. Are the Basic collections certainly not the only kind of data source, we can use with list data binding. alternatively, bind any data structure that implements the ICollection interface or one of its derivatives. The following list sum up many of these data classes:
    - All in-memory collection classes,
        - such as Collection,
        - ArrayList,
        - Hashtable, and
        - Dictionary
    - An ADO.NET DataReader object, which provides
        - connection-based,
        - forward-only, and
        - read-only access to the database.
    - The ADO.NET DataView, which provides a view onto a single disconnected DataTable object.
    - Any other custom object that implements the ICollection interface
- For example, I want to fill a list box with the full names of all the employees contained in the Employees table of the Northwind database.

The Rich Data Controls

- In addition to the simple list controls, ASP.NET includes some rich data controls that support repeated value binding. The rich data controls are entirely a bit different from the simple list controls.

- They are designed only for data binding. They also have the capability to display various properties or fields from each data item, often in a table-based according to a template
- They support higher-level features such as editing and they provide several events that allow us to plug into the control's inner workings at various points.

The List of rich data controls:

1. **GridView:** The GridView is an all-purpose grid control for showing large tables of information. It supports selecting, editing, sorting, and paging. The GridView is the heavyweight of ASP.NET data controls.
2. **DetailsView:** The DetailsView is ideal for displaying a single record at a time, in a table that has one row per field. The DetailsView supports editing and optional paging controls that permit us to browse through an order of records.
3. **FormView:** Like the DetailsView, the FormView displays a single record at a time, supports editing, and provides paging controls for moving through a series of records. The difference is that the FormView is based on templates, which enable to combine fields in a much more flexible way that doesn't need to be based on a table.

Binding to a DataView

- There are few limitations when we bind directly to a DataReader. Because the DataReader is a forward-only cursor, we can't bind data to multiple controls. we also won't have the ability to apply custom sorting and filtering criteria on the fly.
- Finally, unless we take care to program page using generic interfaces such as IDataReader, we lock programs into the data provider we're currently using, making it more difficult to modify or adapt programs in the future. To solve these problems, we can use the disconnected ADO.NET data objects.
- If we fill a disconnected DataSet, we can bind it to one or more controls, and we can tailor the sorting and filtering criteria.
- The DataSet is also completely generic no matter which data provider we use to fill DataSet, the DataSet itself looks the same.
- Never bind directly to a DataSet or DataTable object. Instead, we bind to a DataView object.
- A DataView represents a view of the data in a specific DataTable. That means the following example 4:

Example 4:

```
grid.DataSource = dataTable;
grid.DataBind();
is equivalent to this:
grid.DataSource = dataTable.DefaultView;
grid.DataBind();
```

- Each DataTable includes a default DataView object that's provided through the DataTable.DefaultView property. This skill allows it to bind directly to the DataTable.
- ASP.NET uses the default DataView automatically. The default DataView doesn't apply any sort order and doesn't filter out any rows.
- If we squeeze these settings, we can either configure the default DataView or create our own and explicitly bind it. then use all the sorting and filtering techniques.

12.4 DATA SOURCE CONTROLS – SQLDATASOURCE

- Data source controls, enables avoid writing any data access program.

1. **SqlDataSource:** This data source enables us to connect to any data source that has an ADO.NET data provider. This consists of SQL Server, Oracle, and the OLE DB or ODBC data sources. When using this data source, No need to write the data access program.
2. **ObjectDataSource:** This data source enables us to connect to a custom data access class. This is the preferred approach for large-scale professional web applications.
3. **AccessDataSource:** This data source enables us to read and write the data in an Access database file (.mdb). Access databases do not have a dedicated server engine like SQL Server that coordinates the actions of multiple people and ensures that data is unlikely lost or corrupted. For that reason, Access databases are best suited for very small websites, where few users need to manipulate data at the same time. A much better small-scale data solution is using the free SQL Server Express with the SqlDataSource control.
4. **XmlDataSource:** This data source allows us to connect to an XML file.
5. **SiteMapDataSource:** This data source allows us to connect to the Web.sitemap file that describes the navigational structure of the website.

The Page Life Cycle with Data Binding

- Data source controls can perform two tasks:
  - They can retrieve data from a data source and supply it to linked controls.
  - They can update the data source when edits take place in linked controls.
- In order to know how data controls work, we need to know how they fit into the page life cycle.
- This awareness is important when we run into situations where we need to work with or extend the data binding model. For e.g., to add data or set a selected item in a control after it has been bound to the data source.
- Data binding tasks take place in this order:
  1. The page object is created (based on the .aspx file).
  2. The page life cycle begins, and the Page_init and Page.Load events fire.
  3. All other control events fire.
  4. The data source controls perform any updates. If a row is being updated, the Updating and Updated events fire. If a row is being inserted, the Inserting and Inserted events fire. If a row is being deleted, the Deleting and Deleted events fire.
  5. The Page.PreRender event fires.
  6. The data source controls perform any queries and insert the retrieved data in the linked controls. The Selecting and Selected events fire at this point.
  7. The page is rendered and disposed of.

The SqlDataSource

- Data source controls turn up in the .aspx markup portion of the web page like ordinary controls.
- Example:

  <asp:SqlDataSource ID="SqlDataSource1" runat="server" ... />
- The SqlDataSource acts as a database connection that uses an ADO.NET provider.
- The SqlDataSource needs a generic way to create the Connection, Command, and DataReader objects it requires.
- The only way this is possible is if the data provider comprises a data provider factory.

- The factory has the responsibility of creating the provider specific objects that the SqlDataSource requires in order to access the data source.
- .NET craft with these four provider factories:
  - System.Data.SqlClient
  - System.Data.OracleClient
  - System.Data.OleDb
  - System.Data.Odbc
- These are registered in the machine.config file, and as a result we can use any of them with the SqlDataSource. we select a data source by setting the provider name. Here is a SqlDataSource that connects to a SQL Server database:
  <asp:SqlDataSource ProviderName="System.Data.SqlClient" ... />
- The next step, to supply the required connection string without it, we cannot make any connections.
- All times place it in the <connectionStrings> section of the web.config file to guarantee greater flexibility and ensure not inadvertently changing the connection string, which minimizes the effectiveness of connection pooling.
- Example 5, if we create this connection string:

  Example 5

```
<configuration>
<connectionStrings>
<add name="Northwind"
connectionString="DataSource=localhost;Initial
Catalog=Northwind;
Integrated Security=SSPI"/>
</connectionStrings>
...
</configuration>
```

- SqlDataSource using a $ expression like this:
  <asp:SqlDataSource ConnectionString="<%$ ConnectionStrings:Northwind %>" ... />
- After specifying the provider name and connection string, the next step is to add the query logic that the SqlDataSource will use when it connects to the database.
- Use each SqlDataSource control, created to retrieve a single query.

- Optionally, we can add corresponding commands for deleting, inserting, and updating rows.
- Example, one SqlDataSource is enough to query and update the Customers table in the Northwind database. Need two SqlDataSource controls, if we need to independently retrieve or update Customers and Orders information.
- The SqlDataSource command logic is supplied through four properties:
    - SelectCommand
    - InsertCommand
    - UpdateCommand
    - DeleteCommand,
    - Each command takes a string. The string we supply can be inline SQL
        - SelectCommandType,
        - InsertCommandType,
        - UpdateCommandType,
        - DeleteCommandType
- SqlDataSource that defines a SELECT command for retrieving records. (shown in example 6)

Example 6:

Employees table:

```
<asp:SqlDataSource ID="sourceEmployees" runat="server"
ProviderName="System.Data.SqlClient" ConnectionString="<%$
ConnectionStrings:Northwind %>" SelectCommand="SELECT
EmployeeID, FirstName, LastName, Title, City FROM Employees"/>
```

- Working of Data Source Controls
    1. Select the data source control, and click Refresh Schema in the smart tag. This step triggers the data source control to connect to the database and retrieve the column information for query.
    2. Add a ListBox to form. Set the ListBox.DataSourceID property to the data source control. we can choose it from a drop-down list that shows all the data sources on the form.
    3. Set the ListBox.DataTextField to the column we want to display (in this case, choose EmployeeID). The list of fields should also be provided in a drop-down list.

4. Use the same steps to bind a rich data control. Add a GridView to the web page, and set the GridView.DataSourceID property to the same data source.

5. Run a web page. Don't fret about executing the command or calling DataBind() on the page ASP.NET performs both of those tasks automatically.

Disadvantages of the SqlDataSource

1. **Data access logic embedded in the page:**

   To create a SqlDataSource control, we need to hard-program the SQL statements in the web page.

2. **Maintenance in large applications:**

   Every page that accesses the database needs its own set of SqlDataSource controls. This can turn into a maintenance nightmare, particularly if we have several pages using the same query.

3. **Lack of flexibility:**

   Every data access task requires a separate SqlDataSource. If we want to provide a user with multiple ways to view or query data, this can swamp web pages with data source objects, one for each command variant. This can get complicated fast.

4. **Inapplicability to other data tasks:** The SqlDataSource does not properly represent some types of tasks. The SqlDataSource is planned for data display and data editing scenarios. However, this model divides up if we need to connect to the database and perform another task, such as placing a shipment request into an order pipeline or logging an event.

12.5 SUMMARY

- We have studied data binding expressions and the ASP.NET data source controls in detail.
- The use of GridView, ASP.NET's premier rich data control.
- The main characteristic of data binding is that it's declarative, not programmatic. That means data binding is defined outside the program, alongside the controls in the .aspx page.
- The controls that support single-value data binding allow us to bind some of their properties to a data binding expression.
- Repeated-value binding allows to bind an entire list of information to a control.
- Data source controls, enables avoid writing any data access program.
- Data source controls turn up in the .aspx markup portion of the web page like ordinary controls.

12.6 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

12.7 UNIT END EXERCISES

1. What is data binding? Explain Single-Value Data Binding & Repeated-Value Data Binding?
2. List the Disadvantages of the SqlDataSource
3. Explain the Page Life Cycle with Data Binding.

❄❄❄❄❄❄❄

13

DATA CONTROLS

Unit Structure :

13.0 OBJECTIVE

- To understand ASP.NET's rich data controls, such as the GridView.
- We will take a closer look at the three most powerful data controls.
- To learn how to fine-tune formatting and use features such as selection, sorting, filtering, and templates.
- To learn about advanced scenarios such as showing images, calculating summaries, and creating a master-details list in a single control.

13.1 INTRODUCTION

- In .NET technologie the MultiView is that unlike the rich data controls (the GridView, FormsView, and so on),
- The MultiView is not a naming container. This means that if We add a control named textBox1 to a view, we can’t add another control named textBox1 to another view.
- There’s no real difference between the controls. We add a view and controls in the rest of the page.
- The controls created will be accessible through member variables in page class. This means it’s easy to configure a control in the second view when an event is raised by a control in the first view.
- As a result, the pages created using the MultiView tend to be heavier than normal pages. because the entire control model including the controls from every view is created on every postback and persisted to view state. F

- Example, if We have three views and each view has a different data source control, each time the page is posted back all three data source controls will perform their queries, and every view will be bound, including those that aren't currently visible. To avoid this overhead, We can leave controls unbound and binding them programmatically, else canceling the binding process for views that aren't currently visible.

13.2 GRIDVIEW

- The GridView is an extremely flexible grid control for showing data in a basic grid consisting of rows and columns.
- It includes a wide range of hard-wired features, including selection, paging, sorting, and editing, and it is extensible through templates.
- The great benefit of the GridView over the DataGrid is its support for code-free scenarios.
- Using the GridView, We can achieve many common tasks, such as paging and selection, without writing any code.
- With the DataGrid, We were forced to handle events to implement the same quality.

Defining Columns

- When the property is set, the GridView uses reflection to examine the data object and finds all the fields (of a record) or properties (of a custom object). It then creates a column for each one, in the order that it finds it.
- This automatic column generation is fine for creating quick test pages, but it does not give the flexibility. For example, to hide columns, change their order, or configure some aspect of their display, such as the formatting or heading text. Set AutoGenerateColumns to false and define the columns in the <Columns> section of the GridView control tag.

Different types of column, as shown below The order of column tags find the right-to-left order of columns in the GridView. Shown in Table 1

Column	Description
BoundField	This column displays text from a field in the data source.
ButtonField	This column displays a button for each item in the list.
CheckBoxField	This column displays a check box for each item in the list. It's used automatically for true/false fields (in SQL Server, these are fields that use the bit data type).

Column	Description
CommandField	This column provides selection or editing buttons.
HyperLinkField	This column displays its contents (a field from the data source or static text) as a hyperlink.
ImageField	This column displays image data from a binary field (providing it can be successfully interpreted as a supported image format).
TemplateField	This column allows We to specify multiple fields, custom controls, And arbitrary HTML using a custom template. It gives us the highest degree of control but requires the most work.

Table 1 GridView Column

- BoundField is the most basic column type, which binds to one field in the data object. For example, here's the definition for a single data-bound column that displays the EmployeeID field:
<asp:BoundField DataField="EmployeeID" HeaderText="ID" />
- When first creating a GridView, the AutoGenerateColumns property is not set (and so the default value of true is used). When We bind it to a data source control, nothing changes.
- If We click the Refresh Schema link of the data source control, the AutoGenerateColumns property is flipped to false, and Visual Studio adds a <asp:BoundField> tag for each field it finds in the data source. This approach has several advantages:
    - Easy to use fine-tune column order, column headings, and other details by tweaking the properties of column objects.
    - Can hide columns We don't want to show by removing the column tag.
    - Explicitly defined columns are faster than autogenerated columns. That's because auto generated columns force the GridView to reflect on the data source at runtime.
    - Can add extra columns to the mix for selecting, editing, and more.

Example 1 shows the complete GridView declaration with explicit columns:

Example 1:

```
<asp:GridView ID="gridEmployees" runat="server" DataSourceID="sourceEmployees"
AutoGenerateColumns="False">
```

```
<Columns>
<asp:BoundField DataField="EmployeeID" HeaderText="ID" />
<asp:BoundField DataField="FirstName" HeaderText="First Name" />
<asp:BoundField DataField="LastName" HeaderText="Last Name" />
<asp:BoundField DataField="Title" HeaderText="Title" />
<asp:BoundField DataField="City" HeaderText="City" />
</Columns>
</asp:GridView>
<asp:SqlDataSource ID="sourceEmployees" runat="server"
ConnectionString="<%$ ConnectionStrings:Northwind %>"
ProviderName="System.Data.SqlClient" SelectCommand=
"SELECT EmployeeID, FirstName, LastName, BirthDate, Title, City FROM Employees">
</asp:SqlDataSource>
```

- **Properties:**

  DataField

  This property indicates the name of the field (for a row) or property (for an object) of the data item that We want to display in this column.

  DataFormatString

  This property formats the field. This is useful for getting the right representation of numbers and dates.

  ApplyFormatInEditMode

  If true, the format string will be used to format the value even when it appears in a text box in edit mode. The default is false, which means only the underlying normal will be used.

  HeaderText, FooterText,and HeaderImageUrl

  The first two properties set the text in the header and footer region of the grid, if this grid has a header (GridView.ShowHeader is true) and footer (GridView.ShowFooter is true). The header is most commonly used for a descriptive label such as the field name, while the footer can contain a dynamically calculated value such as a summary. To show an image in the header instead of text, set the HeaderImageUrl property.

ReadOnly

If true, the value for this column can't be changed in edit mode. No edit control will be provided. Primary key fields are often read-only.

InsertVisible

If false, the value for this column can't be set in insert mode. If We want a column value to be set programmatically or based on a default value defined in the database, We can use this feature.

Visible

If false, the column won't be visible in the page (and no HTML will be rendered for it). This property gives We a convenient way to programmatically hide or show specific columns, changing the overall view of the data.

SortExpression

This property specifies an expression that can be appended to a query to perform a sort based on this column. It's used in conjunction with sorting, as described in the "Sorting the GridView" section.

HtmlEncode

If true (the default), all text will be HTML encoded to prevent special characters from mangling the page. You could disable HTML encoding if We want to embed a working HTML tag (such as a hyperlink), but this approach isn't safe. It's always a better idea to use HTML encoding on all values and provide other functionality by reacting to GridView selection events.

NullDisplayText

This property defines the text that will be displayed for a null value. The default is an empty string, although We could change this to a hard- coded value, such as "(not specified)."

ConvertEmptyStringToNull

If this is true, before an edit is committed, all empty strings will be converted to null values.

ControlStyle, HeaderStyle, FooterStyle, and ItemStyle

These properties configure the appearance for just this column, overriding the styles for the row.

- If We don't want to configure columns by hand, select the GridView, and click the ellipsis (...) next to the Columns property in the Properties window. You'll see a Fields dialog box that lets We add, remove, and refine With columns, shown in figure 1

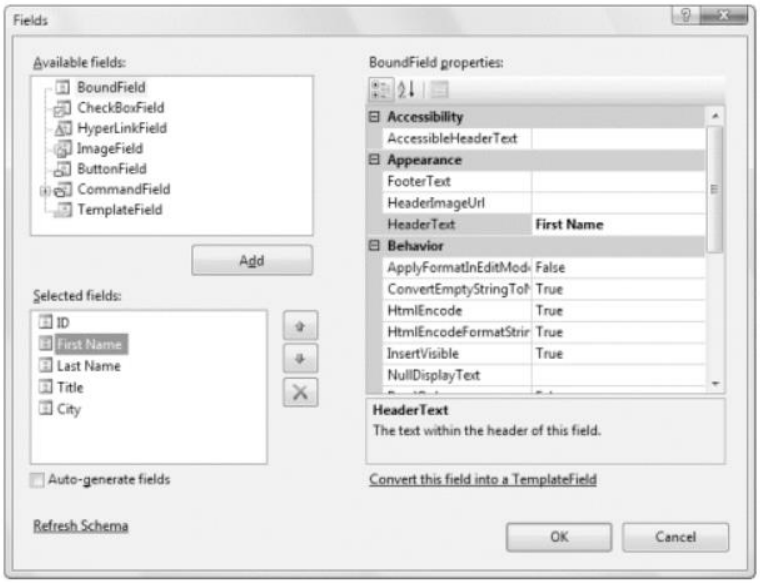


Figure 1. Configuring columns in Visual Studio

In the following sections, implement these topics:

1. Formatting: How to format rows and data values
2. Selecting: How to let users select a row in the GridView and respond accordingly
3. Sorting: How to dynamically reorder the GridView in response to clicks on a column header
4. Paging: How to divide a large result set into multiple pages of data, using both automatic and custom paging code
5. Templates: How to take complete control of laWet, formatting, and editing by defining templates

13.3 DETAILSVIEW

- The GridView and ListView excel at showing dense tables with multiple rows of information. However, sometimes We want to provide a detailed look at a single record.
- ASP.NET includes two controls that are tailored for this purpose: the DetailsView and FormView.
- Both show a single record at a time but can include optional pager buttons that let We step through a series of records. Both support templates, but the FormView requires them. This is the key distinction between the two controls.
- The FormView gives the most flexibility. But if We want to avoid the complexity of templates, the DetailsView gives a simpler model that lets We build a multi row data display out of field objects, in much the same way that the GridView is built out of column objects.

- We can get up to speed with the DetailsView and FormView quite quickly. That's because both the DetailsView and the FormView borrow a portion of the GridView model.

The DetailsView

- The DetailsView is designed to display a single record at a time. It places each piece of information in a separate row of a table.
- The DetailsView can also bind to a collection of items. In this point, it shows the first item in the group.
- It also allows to move from one record to the next using paging controls,
- We can configure the paging controls using the PagingStyle and PagingSettings properties
- The only difference is that there's no support for custom paging, which means the full data source object is always retrieved.
- It's tempting to use the DetailsView pager controls to make a handy record browser. Unfortunately, this approach can be quite inefficient.
- First, a separate postback is required each time the user moves from one record to another. But the real drawback is that each time the page is posted back, the full set of records is retrieved, even though only a single record is shown.
- If We opted to implement a record browser page with the DetailsView, at a bare minimum it must enable caching to reduce the database work .

Example 2: Create a drop-down list and bind this to a data source that queries just the employee names. Then, when a name is selected from the list, retrieve the full details for just that record using another data source. (shown in figure 2)

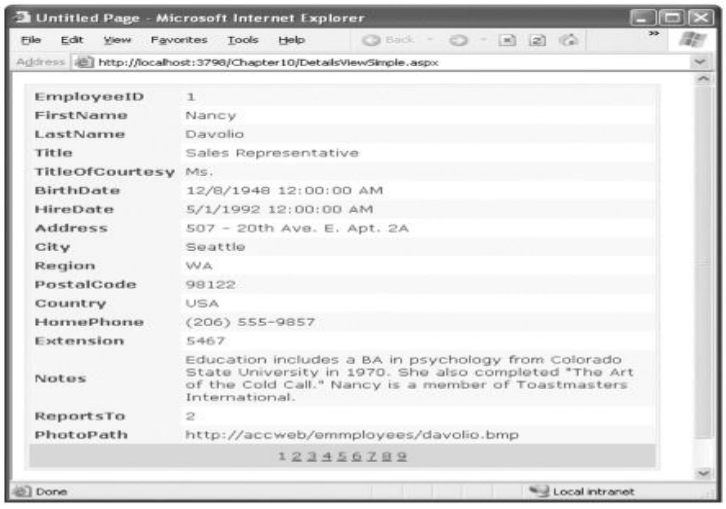


Figure 2. The DetailsView with paging

Defining Fields

- The DetailsView uses reflection to generate the fields it shows.
- It observes the data object and creates a separate field for each field (in a row) or property (in a custom object), just like the GridView.
- To disable this automatic field generation by setting AutoGenerateRows to false. It's then up to us to declare the field objects.

Example 3:

Fields from the data item are represented with the BoundField tag, buttons can be created with the ButtonField, and so on.

Example 3 Field declarations for a DetailsView:

```
<asp:DetailsView ID="DetailsView1" runat="server"
DataSourceID="sourceEmployees"
AutoGenerateRows="False">
<Fields>
<asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID"
/>
<asp:BoundField DataField="FirstName" HeaderText="FirstName" />
<asp:BoundField DataField="LastName" HeaderText="LastName" />
<asp:BoundField DataField="Title" HeaderText="Title" />
<asp:BoundField DataField="TitleOfCourtesy"
HeaderText="TitleOfCourtesy" />
<asp:BoundField DataField="BirthDate" HeaderText="BirthDate" />
...
</Fields>
...
</asp:DetailsView>
```

Record Operations

- The DetailsView holds delete, insert, and edit operations.
- GridView, We don't need to add a CommandField with edit controls. In lieu, We simply set the
  - Boolean AutoGenerateDeleteButton,
  - AutoGenerateEditButton, and
  - AutoGenerateInsertButton properties on the DetailsView control.
- This adds a CommandField at the bottom of the DetailsView with links for these tasks.

- After clicking on the Delete button, the delete operation is performed immediately. However, when We click an Edit or Insert button, the DetailsView changes into edit or insert mode.
- The DetailsView has three modes (as represented by the DetailsViewMode enumeration).
- These modes are ReadOnly, Edit, and Insert.
- Find the current mode at any time by checking the CurrentMode property, and can call ChangeMode() to change it.
- Use the DefaultMode property to create a DetailsView that always begins in edit or insert mode.
- In edit mode, the DetailsView uses standard text box controls just like the GridView. (shown in figure 3)

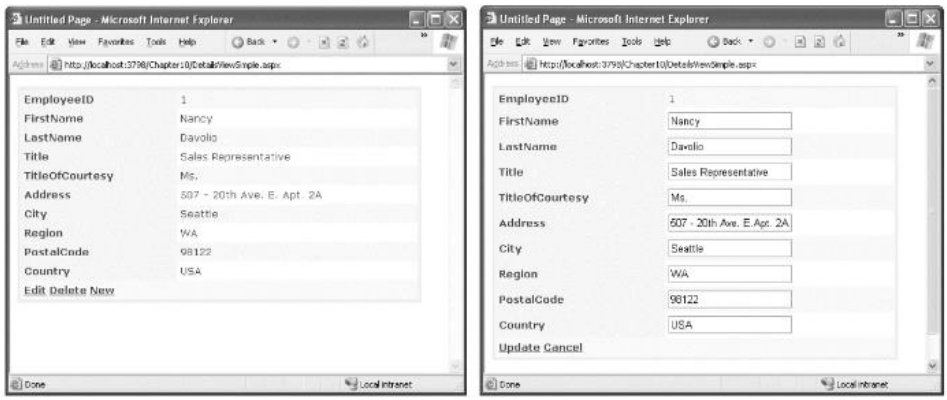


Figure 3. Editing in the DetailsView

13.4 FORMVIEW

- The FormView provides a template-only control for displaying and editing a single record.
- The vision of the FormView template model is that it matches the model of the TemplateField in the GridView quite closely.
- Following are the templates to work with:
  - ItemTemplate
  - EditItemTemplate
  - InsertItemTemplate
  - FooterTemplate
  - HeaderTemplate
  - EmptyDataTemplate
  - PagerTemplate
- This way can take the exact template content which is put in a TemplateField in a GridView (shown in example 4) and place it inside the FormView.

Example 4:

```
<asp:FormView ID="FormView1" runat="server"
DataSourceID="sourceEmployees">
<ItemTemplate>
<b>
<%# Eval("EmployeeID") %> -
<%# Eval("TitleOfCourtesy") %> <%# Eval("FirstName") %>
<%# Eval("LastName") %>
</b>
<hr />
<small><i>
<%# Eval("Address") %><br />
<%# Eval("City") %>, <%# Eval("Country") %>,
<%# Eval("PostalCode") %><br />
<%# Eval("HomePhone") %></i>
<br /><br />
<%# Eval("Notes") %>
<br /><br />
</small>
</ItemTemplate>
</asp:FormView>
```

- The DetailsView, the FormView works in three distinct modes: read-only, insert, and edit.
- The FormView control doesn't support the CommandField class that automatically creates editing buttons.

Table 2 shows the lists of recognized command names:

Command Name	Description	Where It Belongs
Edit	Puts the FormView into edit mode. The FormView renders the current record using the EditItemTemplate with the edit controls We've defined.	The ItemTemplate
Cancel	Cancels the edit or insert operation and returns to the mode specified by the DefaultMode property. Usually, this will be normal mode (FormViewMode.ReadOnly), and the FormView will display the current record using the ItemTemplate.	The EditItemTemplate and InsertItemTemplate
Update	Applies the edit and raises the ItemUpdating and ItemUpdated events on the way.	The EditItemTemplate

Command Name	Description	Where It Belongs
New	Puts the FormView in insertion mode. The FormView displays a new, blank record using the InsertItemTemplate with the edit controls We've defined.	The ItemTemplate
Insert	Inserts the newly supplied data and raises the ItemInserting and ItemInserted events on the way.	The InsertItemTemplate
Delete	Removes the current record from the data source, raising the ItemDeleting and ItemDeleted events. Does not change the FormView mode.	The ItemTemplate

Table 2 List of Command

13.5 SUMMARY

1. We have learned how to build rich data-bound pages.
2. The GridView and considered its support for formatting, selection, sorting, paging, templates, and editing.
3. We considered the template-based ListView and the data controls that are designed to work with a single record at a time: the DetailsView and FormView.
4. We are looking at several common advanced scenarios with data-bound pages.

13.6 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

13.7 UNIT END EXERCISES

1. What is GridView? Explain the List of Column use.
2. What are the different properties used in GridView?
3. What is FormView? Explain the list of recognized commands names?

❄❄❄❄❄❄❄

14

WORKING WITH XML

Unit Structure :

14.0 OBJECTIVE

- To understand the use of XML over the internet
- To study the XML classes.
- To study XMLTextWriter & XMLTextReader classes.

14.1 INTRODUCTION

- XML stands for E**X**tensible **M**arkup **L**anguage XML is a **markup language** much like HTML. XML was designed to **describe data**
- XML tags are not predefined. You must **define your own tags**
- XML shall support a wide variety of applications
- XML is already being used for exchanging information among financial programs, for distributing and updating programs, for writing invoices for delivery over the phones
- XML shall be compatible with SGML
- It shall be easy to write programs which process XML documents.
- XML documents should be human readable and reasonably clear
- The XML should be prepared quickly.
- XML should be easy to create.
- XML uses a **Document Type Definition** (DTD) or an **XML Schema** to describe the data

- XML with a DTD or XML Schema is designed to be **self-descriptive**
- But unlike HTML, XML tags identify the data rather than specify how to display it. Whereas an HTML tag says something like, "Display this data in bold font" (...), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example, <message>...</message>) shown in example 1.

Example 1 of messaging application:

```
<message>
  <to>you@yourAddress.com</to>
  <from>me@myAddress.com</from>
  <subject>XML Is Really Cool</subject>
  <text>
    How many ways is XML cool? Let me count the ways...
  </text>
</message>
```

Advantages of XML

- When XML was first introduced, its success was partly due to its simplicity.
- The rules of XML are much shorter and simpler than the rules of its predecessor, SGML (full form Standard Generalized Markup Language), and simple XML documents are human-readable.
- However, using XML in a professional application isn't simple at all. But if anything, XML is much more useful today than it ever was before.
- The benefits of using XML in a modern application include the following:
    - **Adoption:** XML is ubiquitous. Many companies are using XML to store data or are actively considering it. Whenever data needs to be shared, XML is automatically the first choice that's Inspect.
    - **Extensibility and flexibility:** XML imposes no rules about data semantics and does not tie companies into proprietary networks, unlike EDI (Electronic Data Interchange). XML can fit any type of data and is cheaper to implement.
    - **Related standards and tools:** Another reason for XML's success is the tools (such as parsers) and the surrounding

standards (such as XML Schema, XPath, and XSLT) that help in creating and processing XML documents. As a result, programmers in nearly any language have ready-made components for reading XML, verifying that XML is valid, verifying XML against a set of rules (known as a schema), searching XML, and transforming one format of XML into another.

- XML acts like the glue that allows different systems to work together.
- It helps standardize business processes and transactions between organizations.

14.2 WORKING WITH XML

- XML was designed to store, carry and exchange data. It was not designed to display data.
- An XML application is normally defined by creating a document type definition DTD which is an optional component of an XML document.
- A DTD is like a database schema. It defines and names the elements that can be used in the document, the order in which the elements will appear, the element attributes that can be used etc.
- To use an XML application you usually include its DTD in your XML document. Having the DTD in the document restricts the elements and structure that you can use so that your document is forced to conform to the application standard.
- **Uses of XML in real world**
  - ❖ Storing Databases.
  - ❖ Structuring documents.
  - ❖ Storing Vector Graphics (VML).
  - ❖ Describing Multimedia Applications
  - ❖ Defining channels.
  - ❖ Describing software packages and the interdependencies.
  - ❖ Communicating among applications.
  - ❖ Sending electronic business cards vie e-mai.
  - ❖ XML can Separate Data from HTML.
  - ❖ With XML, your data is stored outside your HTML.
- When HTML is used to display data, the data is stored inside your HTML. With XML, data can be stored in separate XML files. This

way you can concentrate on using HTML for data layout and display, and be sure that changes in the underlying data will not require any changes to your HTML.

- XML data can also be stored inside HTML pages.
- XML is used to Exchange Data
- XML, data can be exchanged between incompatible systems.

XML and B2B

- With the XML, financial information can be exchanged over the Internet.
- XML is going to be the main language for exchanging financial information between businesses over the Internet.
- XML can be used to Share Data
- With XML, plain text files can be used to share data.
- Since XML data is stored in plain text format, XML provides a software- and hardware-independent way of sharing data. This makes it much easier to create data that different applications can work with.
- It also makes it easier to expand or upgrade a system to new operating systems, servers, applications, and new browsers.
- XML can be used to Store Data
- With XML, plain text files can be used to store data.
- XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data.
- XML can make your Data more Useful
- With XML, your data is available to more users.
- XML can be used to Create new Languages
- XML is the mother of WAP and WML. The Wireless Markup Language (WML), used to markup Internet application for handheld devices like mobile phones, is written in XML
- Displaying XML Documents
- Style Sheet Linking
    - You can use a Cascading Style Sheet which is used for HTML or an XSL-eXtensible Style Language which is specifically used for the XML. XSL is more powerful than CSS.

- Data binding
    - This requires you to create an HTML page and link the XML document to it and bind standard HTML elements such as tables to the XML elements.
- Scripting
    - You create an HTML page, link the XML document to it and access and display individual XML by writing script code (Javascript or Vbscript. The browser then exposes the XML document as a Document Object Model- DOM.

14.3 XML CLASSES

- The .NET framework contains a rich set of classes for working with XML data.
- Classes are divided into multiple namespaces.
- The main classes for working with XML data are as follows :
    - XmlTextReader
        - This class provides fast, forward-only access to the raw data contained in an XML file. It parses XML data into tokens, but it does not represent the XML data in an object model such as the W3C XML Document Object Model (DOM).
    - XmlTextWriter
        - This class provides a fast, forward-only method to write data to an XML file. It ensures that the data written to the file conforms to the W3C XML 1.0 standard.
    - XmlDocument
        - This class represents XML using the W3C XML DOM levels 1 and 2. You can use this class to both navigate and edit the nodes in an XML document tree.
    - XmlDataDocument
        - This class enables you to represent both XML and relational data in W3C XML DOM levels 1 and 2. You can use this class with a DataSet to provide both relational and non-relational views of the same underlying data.
    - XmlNodeReader
        - This class provides fast, forward-only access to data represented by the XmlDocument or XmlDataDocument class.
    - DocumentNavigator

- This class enables you to efficiently navigate an XML document represented by the XmlDocument class. It supports the XPath data model for navigation.

- DataDocumentNavigator
  - This class enables you to efficiently navigate an XML document represented by the XmlDataDocument class. It supports the XPath data model for navigation.

- XslTransform
  - This class enables you to transform an XML document with XSL stylesheets. It supports XSLT 1.0 stylesheet syntax.

14.4 XMLTEXTWRITER

- This class represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.
  public class XmlTextWriter : System.Xml.XmlWriter

- XmlTextWriter maintains a namespace stack corresponding to all the namespaces defined in the current element stack. Using XmlTextWriter you can declare namespaces manually which is shown below example 2.

Example 2

```
w.WriteStartElement("root");
w.WriteAttributeString("xmlns", "x", null, "urn:1");
w.WriteStartElement("item","urn:1");
w.WriteEndElement();
w.WriteStartElement("item","urn:1");
w.WriteEndElement();
w.WriteEndElement();
```

- XmlTextWriter promotes the namespace declaration to the root element to avoid having it duplicated on the two child elements. The child elements pick up the prefix from the namespace declaration. Shown in example 3.

Example 3: root elements

```
<root xmlns:x="urn:1">
 <x:item/>
 <x:item/>
</x:root>
```

- XmlTextWriter also allows you to override the current namespace declaration. In the following example, the namespace URI "123" is overridden by "abc" to produce the XML element <x:node xmlns:x="abc"/> Shown in example 4.

Example 4:

```
w.WriteStartElement("x","node","123");
w.WriteAttributeString("xmlns","x",null,"abc");
```

- By using the write methods that take a prefix as an argument you can also specify which prefix to use. In the following example, two different prefixes are mapped to the same namespace URI to produce the XML text <x:root xmlns:x="urn:1"><y:item xmlns:y="urn:1"/></x:root>. Shown in example 5

Example 5:

```
XmlTextWriter w = new XmlTextWriter(Console.Out);
w.WriteStartElement("x","root","urn:1");
 w.WriteStartElement("y","item","urn:1");
 w.WriteEndElement();
w.WriteEndElement();
w.Close();
```

- If namespace conflicts occur, XmlTextWriter resolves them by generating alternate prefixes.(Shown in Table 1) For example, if an attribute and element have the same prefix but different namespaces, XmlWriter generates an alternate prefix for the attribute. The generated prefixes are named n{i} where i is a number beginning at 1. The number is reset to 1 for each element.

Constructors

XmlTextWriter(TextWriter)	Creates an instance of the XmlTextWriter class using the specified TextWriter.
XmlTextWriter(Stream, Encoding)	Creates an instance of the XmlTextWriter class using the specified stream and encoding.
XmlTextWriter(String, Encoding)	Creates an instance of the XmlTextWriter class using the specified file.

Table 1 XmlTextWriter Constructor

- **Properties of** XmlTextWriter shown in table 2

Properties

BaseStream	Gets the underlying stream object.
Formatting	Indicates how the output is formatted.
Indentation	Gets or sets how many IndentChars to write for each level in the hierarchy when Formatting is set to Formatting.Indented.
IndentChar	Gets or sets which character to use for indenting when Formatting is set to Formatting.Indented.
Namespaces	Gets or sets a value indicating whether to do namespace support.
QuoteChar	Gets or sets which character to use to quote attribute values.
Settings	Gets the XmlWriterSettings object used to create this XmlWriter instance.
WriteState	Gets the state of the writer.
XmlLang	Gets the current xml:lang scope.
XmlSpace	Gets an XmlSpace representing the current xml:space scope.

Table 2 XmlTextWriter Properties

- **Methods of** XmlTextWriter shown in table 3

Methods

Close()	Closes this stream and the underlying stream.
Dispose()	Releases all resources used by the current instance of the XmlWriter class.
Dispose(Boolean)	Releases the unmanaged resources used by the XmlWriter and optionally releases the managed resources.
DisposeAsync()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources asynchronously.
DisposeAsyncCore()	Performs application-defined tasks associated with freeing, releasing, or resetting managed resources asynchronously.
Equals(Object)	Determines whether the specified object is equal to the current object.
Flush()	Flushes whatever is in the buffer to the underlying streams and also flushes the underlying stream.

FlushAsync()	Asynchronously flushes whatever is in the buffer to the underlying streams and also flushes the underlying stream.
GetHashCode()	Serves as the default hash function.
GetType()	Gets the Type of the current instance.
LookupPrefix(String)	Returns the closest prefix defined in the current namespace scope for the namespace URI.
MemberwiseClone()	Creates a shallow copy of the current Object.
ToString()	Returns a string that represents the current object.
WriteAttributes(XmlReader, Boolean)	When overridden in a derived class, it writes out all the attributes found at the current position in the XmlReader.
WriteAttributesAsync(XmlReader, Boolean)	Asynchronously writes out all the attributes found at the current position in the XmlReader.
WriteAttributeString(String, String)	When overridden in a derived class, it writes out the attribute with the specified local name and value.

Table 3 Methods of XmlWriter class

14.5 XMLTEXTREADER

- This class represents a reader that provides fast, non-cached, forward-only access to XML data.

  public class XmlTextReader : System.Xml.XmlReader, System.Xml.IXmlLineInfo, System.Xml.IXmlNamespaceResolver

- XmlTextReader provides forward-only, read-only access to a stream of XML data. The current node refers to the node on which the reader is positioned. The reader is advanced using any of the read methods and properties reflect the value of the current node.

- This class implements XmlReader and conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

- XmlTextReader provides the following functionality:
  - Enforces the rules of well-formed XML.
  - XmlTextReader does not provide data validation.
  - Check that DocumentType nodes are well-formed. XmlTextReader checks the DTD for well-formedness, but does not validate using the DTD.

- For nodes where NodeType is XmlNodeType.EntityReference, a single empty EntityReference node is returned (that is, the Value property is String.Empty).

- Does not expand default attributes.
- XmlTextReader does not perform the extra checks required for data validation,
- XmlTextReader provides a fast well-formed parser.
- XmlTextReader throws an XmlException on XML parse errors. After an exception is thrown the state of the reader is not predictable. For example, the reported node type may be different from the actual node type of the current node. Use the ReadState property to check whether a reader is in error state. XmlTextReader Constrctor show in table 4

Constructors

XmlTextReader()	Initializes a new instance of the XmlTextReader.
XmlTextReader(Stream)	Initializes a new instance of the XmlTextReader class with the specified stream.
XmlTextReader(Stream, XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified stream and XmlNameTable.
XmlTextReader(Stream, XmlNodeType, XmlParserContext)	Initializes a new instance of the XmlTextReader class with the specified stream, XmlNodeType, and XmlParserContext.
XmlTextReader(String)	Initializes a new instance of the XmlTextReader class with the specified file.
XmlTextReader(String, Stream)	Initializes a new instance of the XmlTextReader class with the specified URL and stream.
XmlTextReader(String, Stream, XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified URL, stream and XmlNameTable.
XmlTextReader(String, TextReader)	Initializes a new instance of the XmlTextReader class with the

	specified URL and TextReader.
XmlTextReader(String, TextReader, XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified URL, TextReader and XmlNameTable.
XmlTextReader(String, XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified file and XmlNameTable.
XmlTextReader(String, XmlNodeType, XmlParserContext)	Initializes a new instance of the XmlTextReader class with the specified string, XmlNodeType, and XmlParserContext.
XmlTextReader(TextReader)	Initializes a new instance of the XmlTextReader class with the specified TextReader.
XmlTextReader(TextReader, XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified TextReader and XmlNameTable.
XmlTextReader(XmlNameTable)	Initializes a new instance of the XmlTextReader class with the specified XmlNameTable.

Table 4 Constructor for XmlTextReader

Properties (shown in below table 5)

AttributeCount	Gets the number of attributes on the current node.
BaseURI	Gets the base URI of the current node.
CanReadBinaryContent	Gets a value indicating whether the XmlTextReader implements the binary content read methods.
CanReadValueChunk	Gets a value indicating whether the XmlTextReader implements the ReadValueChunk(Char[], Int32, Int32) method.
CanResolveEntity	Gets a value indicating whether this reader can parse and resolve entities.
Depth	Gets the depth of the current node in the XML document.
DtdProcessing	Gets or sets the DtdProcessing enumeration.

Encoding	Gets the encoding of the document.
EntityHandling	Gets or sets a value that specifies how the reader handles entities.
EOF	Gets a value indicating whether the reader is positioned at the end of the stream.
HasAttributes	Gets a value indicating whether the current node has any attributes. (Inherited from XmlReader)
HasValue	Gets a value indicating whether the current node can have a Value other than String.Empty.
IsDefault	Gets a value indicating whether the current node is an attribute that was generated from the default value defined in the DTD or schema.
IsEmptyElement	Gets a value indicating whether the current node is an empty element (for example, <MyElement/>).
Item[Int32]	When overridden in a derived class, gets the value of the attribute with the specified index. (Inherited from XmlReader)
Item[String, String]	When overridden in a derived class, gets the value of the attribute with the specified LocalName and NamespaceURI. (Inherited from XmlReader)
Item[String]	When overridden in a derived class, gets the value of the attribute with the specified Name. (Inherited from XmlReader)
LineNumber	Gets the current line number.
LinePosition	Gets the current line position.
LocalName	Gets the local name of the current node.
Name	Gets the qualified name of the current node.
Namespaces	Gets or sets a value indicating whether to do namespace support.
NamespaceURI	Gets the namespace URI (as defined in the W3C Namespace specification) of the node on which the reader is positioned.
NameTable	Gets the XmlNameTable associated with this implementation.
NodeType	Gets the type of the current node.

Normalization	Gets or sets a value indicating whether to normalize white space and attribute values.
Prefix	Gets the namespace prefix associated with the current node.
ProhibitDtd	Obsolete. Gets or sets a value indicating whether to allow DTD processing. This property is obsolete. Use DtdProcessing instead.
QuoteChar	Gets the quotation mark character used to enclose the value of an attribute node.
ReadState	Gets the state of the reader.
SchemaInfo	Gets the schema information that has been assigned to the current node as a result of schema validation. (Inherited from XmlReader)
Settings	Gets the XmlReaderSettings object used to create this XmlReader instance. (Inherited from XmlReader)
Value	Gets the text value of the current node.
ValueType	Gets The Common Language Runtime (CLR) type for the current node. (Inherited from XmlReader)
WhitespaceHandling	Gets or sets a value that specifies how white space is handled.
XmlLang	Gets the current xml:lang scope.
XmlResolver	Sets the XmlResolver used for resolving DTD references.
XmlSpace	Gets the current xml:space scope.

Table 5 Properties for XmlTextReader

Methods (Shown in table 6)

Close()	Changes the ReadState to Closed.
Dispose()	Releases all resources used by the current instance of the XmlReader class. (Inherited from XmlReader)
Dispose(Boolean)	Releases the unmanaged resources used by the XmlReader and optionally releases the managed resources. (Inherited from XmlReader)

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetAttribute(Int32)	Gets the value of the attribute with the specified index.
GetAttribute(String)	Gets the value of the attribute with the specified name.
GetAttribute(String, String)	Gets the value of the attribute with the specified local name and namespace URI.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetNamespacesInScope (XmlNamespaceScope)	Gets a collection that contains all namespaces currently in-scope.
GetRemainder()	Gets the remainder of the buffered XML.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetValueAsync()	Asynchronously gets the value of the current node. (Inherited from XmlReader)
HasLineInfo()	Gets a value indicating whether the class can return line information.
IsStartElement()	Calls MoveToContent() and tests if the current content node is a start tag or empty element tag. (Inherited from XmlReader)
IsStartElement(String)	Calls MoveToContent() and tests if the current content node is a start tag or empty element tag and if the Name property of the element found matches the given argument. (Inherited from XmlReader)
IsStartElement(String, String)	Calls MoveToContent() and tests if the current content node is a start tag or empty element tag and if the LocalName and NamespaceURI properties of the element found match the given strings. (Inherited from XmlReader)

Table 6 XmlReader Methods

14.6 SUMMARY

- Extensible Markup Language (XML) is a markup language used to describe the content and structure of data in a document.
- It is a simplified version of Standard Generalized Markup Language (SGML).
- XML is an industry standard for delivering content on the Internet. Because it provides a facility to define new tags, XML is also extensible.
- Like HTML, XML uses tags to describe content. However, rather than focusing on the presentation of content, the tags in XML describe the meaning and hierarchical structure of data. This functionality allows for the sophisticated data types that are required for efficient data interchange between different programs and systems. Further, because XML enables separation of content and presentation, the content, or data, is portable across heterogeneous systems.

14.7 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

14.8 UNIT END EXERCISES

1. What is XML? Explain with examples.
2. What are the advantages of XML?
3. Explain the different types of XML Classes?
4. Write a short note on:

a. XMLTextWriter

b. XMLTextReader

❄❄❄❄❄❄❄

15

CACHING

Unit Structure :

15.0 OBJECTIVE

- To understand the use of caching in .NET technologies.
- To study when to use caching.
- To learn the output and data catching techniques.

15.1 INTRODUCTION

- Caching is the technique of storing a memory copy of some information that's costly to create.
- Example, cache the results of a complex query so that subsequent requests don't need to access the database at all. Rather, they can snatch the appropriate object directly from server memory, a much faster proposition.
- The real picture of caching is that unlike many other performance enhancing techniques, caching bolsters both performance and scalability.
- Performance is better because the time taken to retrieve the information is cut down dramatically.
- Scalability is improved because you work around bottlenecks such as database connections.
- Server memory is a limited resource; if you try to store too much, some of that information will be paged to disk, potentially slowing down the entire system. That's why the best coaching strategies are self-limiting.

- When we store information in a cache, we expect to find it there on a future request most of the time. However, the lifetime of that information is at the discretion of the server.
- If the cache becomes full or other applications consume a large amount of memory, information will be selectively evicted from the cache, ensuring that performance is maintained. It's this self-sufficiency that makes caching so powerful.
- Cache can be completely rendered HTML for a page, a portion of that HTML, or arbitrary objects. We can customize expiration policies and set up dependencies so that items are automatically removed when other resources such as files or database tables are modified.

15.2 WHEN TO USE CACHING

- Caches are usually used to keep track of persistent responses to user requests.
- It can also be used in the case of storing results of long computational operations.
- Caching is storing data in a location different from the main data source such that it's faster to access the data. Like load balancers, caching can be used in various places of the system.
- Caching is done to avoid redoing the same complex computation repeatedly.
- Caching is used to improve the time complexity of algorithms for example, say dynamic programming, the memorization technique to reduce time complexity.
- In the case of system design concepts, caching as a concept is also the same.
- Caching is used to improve the speed of a system.
- To improve the latency of a system, we need to use caching.
- To reduce network requests can also be a cause for using caching.
- Caching provides a twofold, threefold, or even tenfold performance improvement by retaining important data for just a short period of time.
- ASP.NET really has two types of caching:
    - **Output caching:**
        - This is the simplest type of caching.
        - It stores a copy of the final rendered HTML page that is sent to the client. The next client that submits a request

for this page doesn't actually run the page. Instead, the final HTML output is sent automatically.

- The time that would have been required to run the page and its code is completely reclaimed.

- **Data caching:**
  - This type of caching is carried out manually in code.
  - To use data caching, we store important pieces of information that are time consuming to reconstruct in the cache. Other pages can check for the existence of this information and use it, thereby bypassing the steps ordinarily required to retrieve it.
  - Data caching is conceptually the same as using application state, but it's much more server-friendly because items will be removed from the cache automatically when it grows too large and performance could be affected. Items can also be set to expire automatically.

- **Fragment caching:**
  - This is a specialized type of output caching instead of caching the HTML for the whole page, it allows you to cache the HTML for a portion of it.
  - Fragment caching works by storing the rendered HTML output of a user control on a page.
  - The next time the page is executed, the same page events fire, but the code for the appropriate user control is not executed.

- Data **source caching:**
  - This is the caching that's built into the data source controls, including the SqlDataSource, ObjectDataSource, and XmlDataSource. The data source caching uses data caching.
  - The difference is that you don't need to handle the process explicitly. Instead, you simply configure the appropriate properties, and the data source control manages the caching storage and retrieval.

15.3 OUTPUT CACHING

- With this type of caching, the final rendered HTML of the page is cached.
- When the same page is requested again, the control objects are not created, the page life cycle doesn't start, and none of your code executes. Instead, the cached HTML is served. Clearly, output

caching gets the theoretical maximum performance increase, because all the overhead of your code is sidestepped.

1. Declarative Output Caching

Example 1 : create a simple page that displays the current time of day. The code for this page is straightforward. It simply sets the date to appear in a label when the Page.Load event occurs:

Example 1:

```
protected void Page_Load(Object sender, EventArgs e)
{
lblDate.Text = "The time is now:<br />";
lblDate.Text += DateTime.Now.ToString();
}
```

Two ways to add this page to the output cache. The most common approach is to insert the OutputCache directive at the top of your .aspx file, just below the Page directive shown in example 2:

Example 2:

```
<%@ OutputCache Duration="20" VaryByParam="None" %>
```

2. Caching and the Query String

- Caching is deciding when a page can be reused and when information must be accurate up to the latest second. Use caching to efficiently reuse slightly stale data without a problem, and with a considerable performance improvement.
- Occasionally information needs to be dynamic. One example is if the page uses information from the current user's session to tailor the user interface. In this type, full page caching just isn't suitable. Second example is if the page is receiving information from another page through the query string.
- ASP.NET provides options. We can set the VaryByParam attribute to * to indicate that the page uses the query string and to instruct ASP.NET to cache separate copies of the page for different query string arguments, as shown below:

  <%@ OutputCache Duration="20" VaryByParam="*" %>

- When requesting the page with additional query string information, ASP.NET will inspect the query string. If the string matches a previous request, and a cached copy of that page exists, it will be reused.
- To get a better idea how this process works, consider the following series of requests:
    1. You request a page without any query string parameter and receive page copy A.
    2. You request the page with the parameter ProductID=1. You receive page copy B.
    3. Another user requests the page with the parameter ProductID=2. That user receives copy C.
    4. Another user requests the page with ProductID=1. If the cached output B has not expired, it's sent to the user.
    5. The user then requests the page with no query string parameters. If copy A has not expired, it's sent from the cache.

Caching with Specific Query String Parameters

- Setting VaryByParam="*" allows you to use caching with dynamic pages that vary their output based on the query string.
- This approach could be extremely useful for a product detail page, which receives a product ID in its query string. With vary-by-parameter caching, you could store a separate page for each product, thereby saving a trip to the database. However, to gain performance benefits you might have to increase the cached output lifetime to several minutes or longer.
- Pages that accept a wide range of different query string parameters just are not suited to output caching.
- Setting VaryByParam to the wildcard asterisk (*) is unnecessarily vague. It's usually better to specifically identify an important query string variable by name.

  example:

  <%@ OutputCache Duration="20" VaryByParam="ProductID" %>
- ASP.NET will examine the query string looking for the ProductID parameter. Requests with different ProductID parameters will be coached separately, but all other parameters will be ignored. This is particularly useful if the page may be passed additional query string information that it doesn't use. ASP.NET has no way to distinguish the "important" query string parameters without your help. Specify several parameters, as long as you separate them with semicolons, as follows:

  <%@ OutputCache Duration="20"
  VaryByParam="ProductID;CurrencyType" %>

In this case, the query string will cache separate versions, provided the query string differs by ProductID or CurrencyType.

Custom Caching Control

- ASP.NET allows you to create your own procedure that decides whether to cache a new page version or reuse an existing one.
- This code examines whatever information is appropriate and then returns a string.
- ASP.NET uses this string to implement caching. If your code generates the same string for different requests, ASP.NET will reuse the cached page.
- If your code generates a new string value, ASP.NET will generate a new cached version and store it separately.
- One way you could use custom caching is to cache different versions of a page based on the browser type.
- The following example uses the name browser because pages will be cached based on the client browser:

  <%@ OutputCache Duration="10" VaryByParam="None" VaryByCustom="browser" %>

- To create the procedure that will generate the custom caching string. This procedure must be coded in the global.asax application file, as shown below example 3:

Example 3

```
public override string GetVaryByCustomString(
HttpContext context, string arg)
{
// Check for the requested type of caching.
if (arg == "browser")
{
// Determine the current browser.
string browserName;
browserName = Context.Request.Browser.Browser;
browserName                                                    +=
Context.Request.Browser.MajorVersion.ToString();
// Indicate that this string should be used to vary caching.
return browserName;
}
else
{
return base.GetVaryByCustomString(context, arg);
}
}
```

- The GetVaryByCustomString() function passes the VaryByCustom name in the arg parameter. This allows you to create an application that implements several types of custom caching in the same function. Each different type would use a different VaryByCustom name (such as Browser, BrowserVersion, or DayOfWeek). Your GetVaryByCustomString() function would examine the VaryByCustom name and then return the appropriate caching string. If the caching strings for different requests match, ASP.NET will reuse the cached copy of the page.

- The OutputCache directive also has a third attribute that you can use to define caching. This attribute, VaryByHeader, allows you to store separate versions of a page based on the value of an HTTP header received with the request. You can specify a single header or a list of headers separated by semicolons.

- We can use this technique with multilingual sites to cache different versions of a page based on the client browser language, as follows:

  <%@ OutputCache Duration="20" VaryByParam="None" VaryByHeader="Accept-Language" %>

Caching with the HttpCachePolicy Class

- Cache property, which provides an instance of the System.Web.HttpCachePolicy class.

- This object provides properties that allow you to turn on caching for the current page.

- This allows you to decide programmatically whether you want to enable output caching.

- Example 4 shows, the date page has been rewritten so that it automatically enables caching when the page is first loaded. This code enables caching with the SetCacheability() method, which specifies that the page will be cached on the server and that any other client can use the cached copy of the page. The SetExpires() method defines the expiration date for the page, which is set to be the current time plus 60 seconds.

Example 4

```
protected void Page_Load(Object sender, EventArgs e)
{
// Cache this page on the server.
Response.Cache.SetCacheability(HttpCacheability.Public);
// Use the cached copy of this page for the next 60 seconds.
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
// This additional line ensures that the browser can't
```

```
// invalidate the page when the user clicks the Refresh button
// (which some rogue browsers attempt to do).
Response.Cache.SetValidUntilExpires(true);
lblDate.Text = "The time is now:<br />" + DateTime.Now.ToString();
}
```

Post-Cache Substitution and Fragment Caching

- Fragment caching:
  - In this case, you identify just the content you want to cache, wrap that in a dedicated user control, and cache just the output from that control.
- Post-cache substitution:
  - In this case, you identify just the dynamic content you don't want to cache. You then replace this content with something else using the Substitution control.
- Fragment caching is the easiest to implement.
- If you have a small, distinct portion of content to cache, fragment caching makes the most sense.
- Conversely, if you have only a small bit of dynamic content, post-cache substitution may be the more straightforward approach. Both approaches offer similar performance shown in below example 5.

Example 5

```
private static string GetDate(HttpContext context)
{
return "<b>" + DateTime.Now.ToString() + "</b>";
}
```

To get this in the page, you need to use the Response.WriteSubstitution() method at some point:

```
protected void Page_Load(object sender, EventArgs e)
{
Response.Write("This date is cached with the page: ");
Response.Write(DateTime.Now.ToString() + "<br />");
Response.Write("This date is not: ");
Response.WriteSubstitution(new HttpResponseSubstitutionCallback(GetDate));
}
```

15.4 DATA CACHING

- Data caching is the most flexible type of caching, but it also forces you to take specific additional steps in your code to implement it.
- The basic principle of data caching is that you add items that are expensive to create a special built-in collection object (called Cache).
- This object works much like the Application object. It's globally available to all requests from all clients in the application. However, a few key differences exist:
    - **The Cache object is thread-safe:** This means you don't need to explicitly lock or unlock the Cache collection before adding or removing an item.
    - **Items in the cache are removed automatically:** ASP.NET will remove an item if it expires, if one of the objects or files it depends on is changed, or if the server becomes low on memory. This means you can freely use the cache without worrying about wasting valuable server memory, because ASP.NET will remove items as needed. But because items in the cache can be removed, you always need to check if a cached object exists before you attempt to use it. Otherwise, you'll run into a NullReferenceException.
    - **Items in the cache support dependencies:** We can link a cached object to a file, a database table, or another type of resource. If this resource changes, your cached object is automatically deemed invalid and released.
    - **Adding Items to the Cache** As with the Application and Session collections, you can add an item to the Cache collection just by assigning a new key name:

      Cache["key"] = item;
    - A better approach is to use the Cache.Insert() method. Table 1 below lists the four versions of the Insert() method.

Overload	Description
Cache.Insert(key, value)	Inserts an item into the cache under the specified key name, using the default priority and expiration. This is the same as using the indexer- based collection syntax and assigning to a new key name.
Cache.Insert(key, value, dependencies)	Inserts an item into the cache under the specified key name, using the default priority and expiration. The last parameter contains a CacheDependency object that links to other files or cached items and allows the cached

Overload	Description
	item to be invalidated when these change.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration)	Inserts an item into the cache under the specified key name, using the default priority and the indicated sliding or absolute expiration policy. This is the most commonly used version of the Insert() method.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration, priority, onRemoveCallback)	Allows you to configure every aspect of the cache policy for the item, including expiration, priority, and dependencies. In addition, you can submit a delegate that points to a method you want invoked when the item is removed.

Table 1 Insert Methods

Example that stores an item with a sliding expiration policy of 10 minutes, with no dependencies:

Cache.Insert("MyItem", obj, null,

DateTime.MaxValue, TimeSpan.FromMinutes(10));

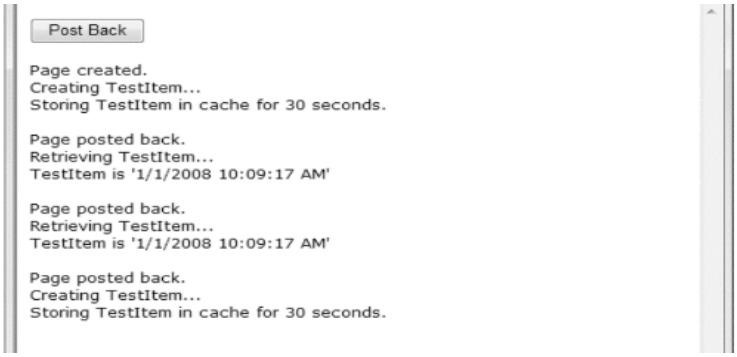
A Simple Cache Test

The following Example 6 presents a simple caching test. An item is cached for 30 seconds and reused for requests in that time.

Example 6:

```
protected void Page_Load(Object sender, EventArgs e)
{
if (this.IsPostBack)
{
lblInfo.Text += "Page posted back.<br />";
}
else
{
lblInfo.Text += "Page created.<br />";
}
DateTime? testItem = (DateTime?)Cache["TestItem"];
if (testItem == null)
{
lblInfo.Text += "Creating TestItem...<br />";
testItem = DateTime.Now;
lblInfo.Text += "Storing TestItem in cache ";
lblInfo.Text += "for 30 seconds.<br />";
```

```
Cache.Insert("TestItem", testItem, null,
DateTime.Now.AddSeconds(30), TimeSpan.Zero);
}
else
{
lblInfo.Text += "Retrieving TestItem...<br />";
lblInfo.Text += "TestItem is '" + testItem.ToString();
lblInfo.Text += "'<br />";
}
lblInfo.Text += "<br />";
}
```



Cache Priorities

- Setting a priority while adding an item to the cache. The priority only has an effect if ASP.NET needs to perform cache searches, which is the process of removing cached items early because memory is becoming scarce.
- To assign a cache priority, choose a value from the CacheItemPriority enumeration shown in table 2.

Lists of all the value:

Value	Description
High	These items are the least likely to be deleted from the cache as the server frees system memory.
AboveNormal	These items are less likely to be deleted than Normal priority items.
Normal	These items have the default priority level. They are deleted only after Low or BelowNormal priority items have been removed.
BelowNormal	These items are more likely to be deleted than Normal priority items.
Low	These items are the most likely to be deleted from

	the cache as the server frees system memory.
NotRemovable	These items will ordinarily not be deleted from the cache as the server frees system memory.

Table 2 Cache properties value

Caching with the Data Source Controls

- The SqlDataSource, ObjectDataSource, and XmlDataSource all support built-in data caching.
- Using caching with these controls is highly recommended, because the data source controls often generate extra query requests.
- For example, they requery after every postback when parameters change, and they perform a separate query for every bound control, even if those controls use exactly the same command, Even a little caching can reduce this overhead.

To support caching, the data source controls all use the same properties, which are listed below

Cache-Related Properties of the Data Source Controls shown in table 3

Property	**Description**
EnableCaching	If true, caching is switched on. It's false by default.
CacheExpirationPolicy	Uses a value from the DataSourceCacheExpiry enumeration— Absolute for absolute expiration (which times out after a fixed interval of time) or Sliding for sliding expiration (which resets the time window every time the data object is retrieved from the cache).
CacheDuration	The number of seconds to cache the data object. If you are using sliding expiration, the time limit is reset every time the object is retrieved from the cache. The default value, 0 (or Infinite), keeps cached items perpetually.
CacheKeyDependency and SqlCacheDependency	Allows you to make a cached item dependent on another item in the data cache (CacheKeyDependency) or on a

	table in your database (SqlCacheDependency). Dependencies are discussed in the "Cache Dependencies" section.

Table 3 Cache-Related Properties

15.5 SUMMARY

- Caching is the technique of storing a memory copy of some information that's costly to create.
- Caches are usually used to keep track of persistent responses to user requests.
- Caching provides a twofold, threefold, or even tenfold performance improvement by retaining important data for just a short period of time.
- Data caching is the most flexible type of caching, but it also forces you to take specific additional steps in your code to implement it.
- Setting a priority while adding an item to the cache.

15.6 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

15.7 UNIT END EXERCISES

1. What is caching? When to Use Caching?
2. Explain Output Caching with an example?
3. What is Data Caching? Explain a few key differences?
4. Write a short note on custom caching control?

❄❄❄❄❄❄❄

16

LINQ

Unit Structure :

16.0 OBJECTIVE

- This chapter would make you understand the following concepts:
  - LINQ
  - LINQ operators
  - Data model to an object model
  - LINQ queries
- The learner can use LINQ to query multiple data sources.
- The learners can learn different LINQ operator concept to write different queries against different databases.
- Learners can use the same basic query expression patterns to query and transform data into SQL databases, ADO.NET datasets, XML documents, and .NET collections and flows.

16.1 INTRODUCTION

- The data required for an application can be stored in a relational database, business objects, XML file, or on web services.
- Retrieving in memory objects is simple and less expensive than retrieving data from a database or XML file. The data accessed are not used directly. It required ordering, grouping and altering.
- LINQ allows joining data from different data sources and executing set data processing operations in a few lines of code. It defines a common syntax and a programming model to query different types of data sources using a common language.
- LINQ is a deeply integrated part of .NET and the C# language. it can be used equally well in any type of .NET application, most likely to use LINQ as part of a database component. we can use LINQ in addition to ADO.NET data access code.

Advantages of LINQ

- Familiar language: Developers don't have to learn a new query language for each type of data source or data format.
- Less coding: It reduces the amount of code to be written as compared with a more traditional approach.
- Readable code: LINQ makes the code more readable so other developers can easily understand and maintain it.
- Standardized way of querying multiple data sources: The same LINQ syntax can be used to query multiple data sources.
- Compile time safety of queries: It provides type checking of objects at compile time.
- IntelliSense Support: LINQ provides IntelliSense for generic collections.
- Shaping data: You can retrieve data in different shapes.

Disadvantages of LINQ are:

- With the use of LINQ, it's very difficult to write a complex query like SQL.
- It was written in the code, and we cannot make use of the Cache Execution plan, which is the SQL feature as we do in the stored procedure.
- If the query is not written correctly, then the performance will be degraded.
- If we make some changes to our queries, then we need to recompile the application and need to redeploy the dll to the server.

16.2 UNDERSTANDING LINQ

Operators:

- A set of extension methods forming a query pattern is known as LINQ Standard Query Operators. As building blocks of LINQ query expressions, these operators offer a range of query capabilities like filtering, sorting, projection, aggregation, etc. LINQ standard query operators can be categorized into the following ones on the basis of their Functionality. Shown in Table 1.

Operators	Description
Select	it is to fields of the return statement
From	It is to specify the data source for the query
Where	It is to specify the condition while retrieving the data
OrderBy	It is to specify the field by which the data is sorted
GroupBy	It is to specify the field by which the data is grouped
Sum, Min, Max, Average, Count	These are the aggregation operators in LINQ. They allows to perform mathematical calculations on the objects in the result set.

Take	It specifies how many rows we require in output from the start position of the data source but when we define a criteria in it then criteria is evaluated first before the start position is determined.
Skip	It bypasses a specified number of contiguous rows from a data source and returns the remaining rows from the data source. It can skip rows from the top or it can also skip rows depending on a certain criteria.
TakeWhile	It returns the sequence of elements starting from the beginning of data source until the mentioned condition is false.
SkipWhile	It gives the result reverse of TakeWhile operator. It skips the number of elements in collection until the condition is true.
Join	It is similar to a SQL INNER JOIN. It only gives the result when it finds a match between two data sources.

Table 1 List of Operators

Implementations:

LINQ includes three basic types of implementation

a. LINQ to objects: It used to query almost any kind of collections that exists in the .NET Framework.

b. LINQ to XML: It used to query XML data directly in your web application

c. LINQ to ADO.NET: It used to access data from SQL Server and many other different kinds of data sources. Using LINQ to ADO.NET you can perform LINQ to DataSet, LINQ to SQL, and LINQ to Entities. LINQ to SQL allows writing object-oriented queries against SQL Server databases. LINQ to DataSet allows writing queries against the DataSet. LINQ to Entities allows writing queries against the ADO.net Entity framework.

16.3 LINQ BASICS

LINQ includes three basic types of implementation

a. LINQ to objects: It used to query almost any kind of collections that exist in the .NET Framework.

b. LINQ to XML: It used to query XML data directly in your web application

c. LINQ to ADO.NET: It used to access data from SQL Server and many other different kinds of data sources. Using LINQ to ADO.NET you can perform LINQ to DataSet, LINQ to SQL, and LINQ to Entities. LINQ to SQL allows writing object-oriented

queries against SQL Server databases. LINQ to DataSet allows writing queries against the DataSet. LINQ to Entities allows writing queries against the ADO.net Entity framework.

Mapping Your Data Model To An Object Model

Using LINQ to Objects

- The term LINQ to Objects refers to the use of LINQ queries to access in memory data structures. Query any type that supports IEnumerable<T>.
- This means that you can use LINQ queries not only with user-defined lists, arrays, dictionaries, and so on, but also in conjunction with .NET Framework APIs that return collections.
- For example, The **System.Reflection** classes return information about types stored in a specified assembly, and then filter those results using LINQ.
- LINQ queries offer three main advantages over traditional foreach loops:
    - They are more concise and readable, especially when filtering multiple conditions.
    - They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
    - They can be ported to other data sources with little or no modification.

LINQ to XML:

Provides creation and manipulation of XML documents using the same syntax and general query mechanism as the other LINQ variety shown in example 1.

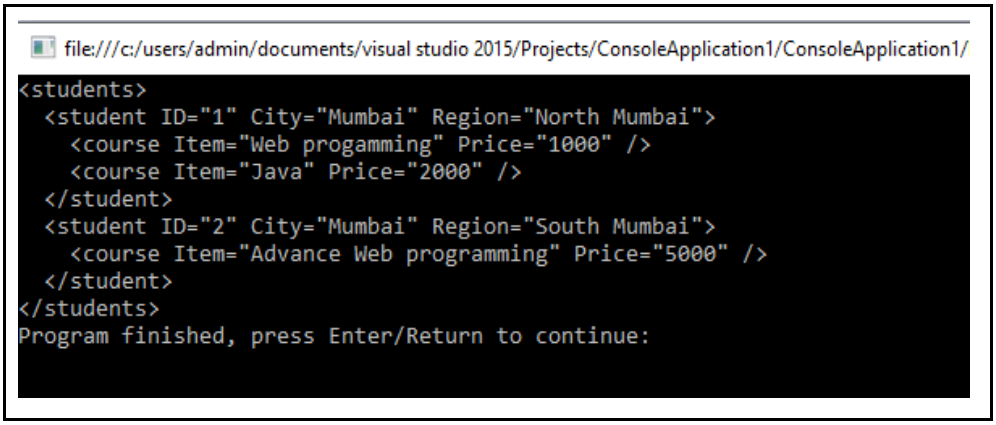
Example 1: LINQ to create XML files to store student details.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
namespace ConsoleApplication1
{
class Program
{
```

```
static void Main(string[] args)
{
XDocument xdoc = new XDocument(
new XElement("students",
new XElement("student",
new XAttribute("ID", "1"),
new XAttribute("City", "Mumbai"),
new XAttribute("Region", "North Mumbai"),
new XElement("course",
new XAttribute("Item", "Web progamming"),
new XAttribute("Price", 1000)
),
new XElement("course",
new XAttribute("Item", "Java"),
new XAttribute("Price", 2000)
)
),
new XElement("student",
new XAttribute("ID", "2"),
new XAttribute("City", "Mumbai"),
new XAttribute("Region", "South Mumbai"),
new XElement("course",
new XAttribute("Item", "ADO.NET"),
new XAttribute("Price", 5000)
)
)
)
);
Console.WriteLine(xdoc);
Console.Write("Program finished, press Enter/Return to
continue:");
Console.ReadLine();
}
}
}
```

Output:



LINQ to ADO.net

- LINQ to ADO.Net means using the LINQ queries on the objects in the ADO.Net.
- The LINQ to ADO.Net gives us a chance to write the LINQ Queries on Enumerable objects in ADO.Net and the LINQ to ADO.Net is having the three types of LINQ technologies available.
- These are LINQ to Dataset, LINQ to SQL and LINQ to Entities.

LINQ Works

- To use LINQ, you create a LINQ expression.
- The return value of a LINQ expression is an iterator object that implements IEnumerable<T>.
- When you enumerate over the iterator object, LINQ performs its work.
- There are three separate ADO.NET Language-Integrated Query (LINQ) technologies:
  - LINQ to DataSet
  - LINQ to SQL, and
  - LINQ to Entities.
  - LINQ to DataSet provides richer, optimized querying over the DataSet and LINQ to SQL enables you to directly query SQL Server database schemas, and LINQ to Entities allows you to query an Entity Data Model.
- The following figure 1 provides an overview of how the ADO.NET LINQ technologies relate to high-level programming languages and LINQ-enabled data sources.

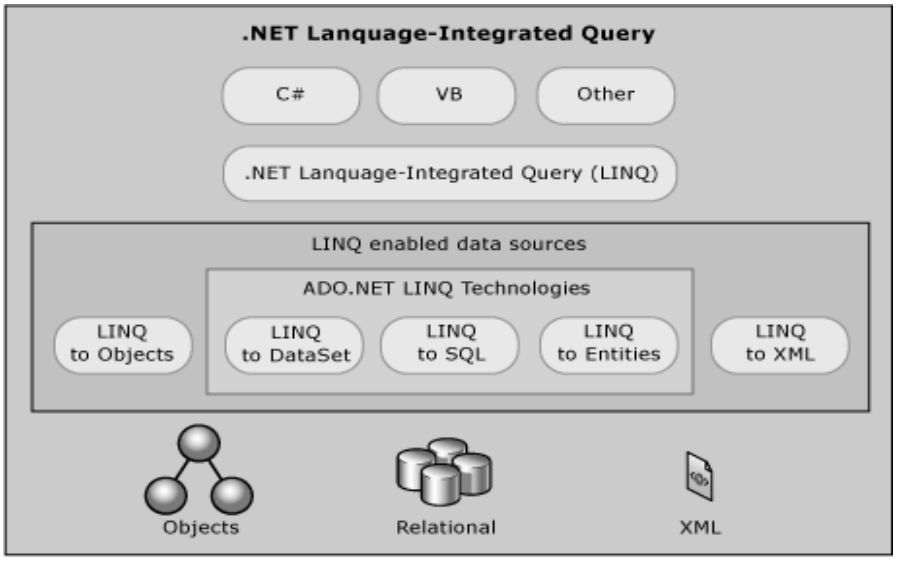


Figure 1. ADO.NET LINQ technologies

LINQ to DataSet

- The DataSet is a key element of the disconnected programming model that ADO.NET is built on, and is widely used.
- LINQ to DataSet enables developers to build richer query capabilities into DataSet by using the same query formulation mechanism that is available for many other data sources.

LINQ to SQL

- LINQ to SQL is a useful tool for developers who do not require mapping to a conceptual model.
- By using LINQ to SQL, use the LINQ programming model directly over existing database schema.
- LINQ to SQL enables developers to generate .NET Framework classes that represent data. Rather than mapping to a conceptual data model, these generated classes map directly to database tables, views, stored procedures, and user-defined functions.

LINQ to Entities

- Most applications are currently written on top of relational databases. At some point, these applications will need to interact with the data represented in a relational form.
- Database schemas are not always ideal for building applications, and the conceptual models of application are not the same as the logical models of databases.
- The Entity Data Model is a conceptual data model that can be used to model the data of a particular domain so that applications can interact with data as objects. For more information, see ADO.NET Entity Framework.

Aggregate functions (shown in example 2)

Aggregate functions are extension methods in LINQ. The following are the Aggregate functions.

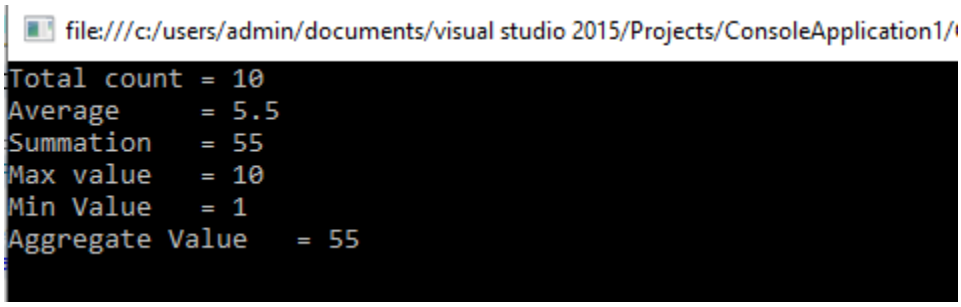
- Average(): It is used to average the number of elements in the list or collection.
- Count(): COUNT () function in LINQ is used to count the number of elements in the list or collection.
- Max(): Max () function in LINQ is used to return the maximum value from the collection.
- Min(): MIN () function of LINQ is useful to get the minimum value from a collection or list.
- Sum(): In LINQ sum() function is used to calculate the sum of the items in collections or lists.
- Aggregate: Aggregate() function is used to perform the operation on each item of the list. The Aggregate() function performs the action on the first and second elements and then carries forward the result.

Example 2: LINQ with Aggregate functions.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
int[] n = { 1,2,3,4,5,6,7,8,9,10 };
Console.WriteLine("Total count = " + n.Count());
Console.WriteLine("Average = " + n.Average());
Console.WriteLine("Summation = " + n.Sum());
Console.WriteLine("Max value = " + n.Max());
Console.WriteLine("Min Value = " + n.Min());
Console.WriteLine("Aggregate Value = " + n.Aggregate((a, b) =>
a + b));
Console.ReadLine();
```

```
}
}
}
```

Output:



LINQ Sorting Operators (Shown in Example 3)

- Sorting Operators in LINQ are used to change the order or sequence of the data (either ascending or descending), which is based on one or more attributes.
    - **OrderBy:** This operator will sort the values in ascending order.
    - **OrderByDescending:** This operator will sort the values in descending order.
    - **ThenBy:** This operator is used to perform the secondary sorting in ascending order.
    - **ThenByDescending:** This operator is used to perform the sorting in descending order.
    - **Reverse:** This operator is used to reverse the order of elements in the collection.

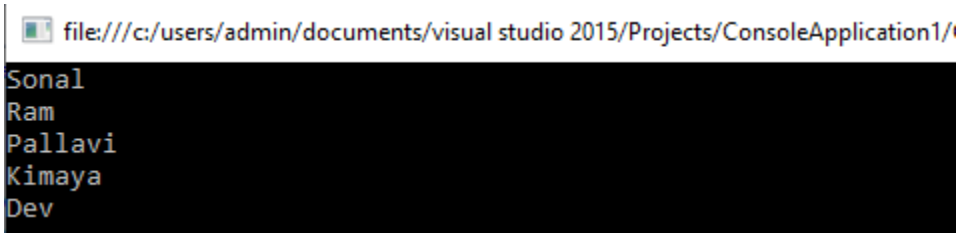
Example 3: LINQ to sort employee list by employee name in descending order.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
namespace ConsoleApplication1
{
class Employee
{
```

```
public int EmployeeID { get; set; }
public string EmpName { get; set; }
public int Age { get; set; }
}
class Program
{
static void Main(string[] args)
{
List<Employee> empList = new List<Employee>()
{
new Employee() { EmployeeID = 1, EmpName = "Pallavi", Age = 28 } ,
new Employee() { EmployeeID = 2, EmpName = "Kimaya", Age = 26 } ,
new Employee() { EmployeeID = 3, EmpName = "Dev", Age = 35 } ,
new Employee() { EmployeeID = 4, EmpName = "Ram" , Age = 25 } ,
new Employee() { EmployeeID = 5, EmpName = "Sonal" , Age = 28 }
};
var empResult = empList.OrderByDescending(x => x.EmpName);
foreach (var emp in empResult)
{
Console.WriteLine(emp.EmpName);
}
Console.ReadLine();
}
}
}
```

Output:

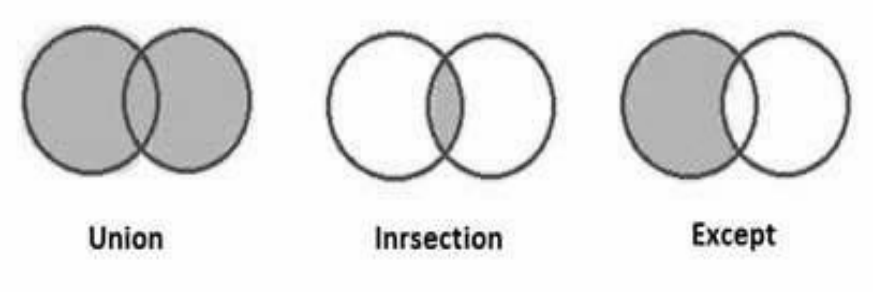


LINQ SET OPERATORS

LINQ provides standard set operators such as Except, Intersect, Union, and Distinct. These operators behave in the same way as in DBMS.

- **Union operator :** This operator requires two collections of items. After applying union operator, you will get a new collection. It removes the duplicate entries.
- **Intersect operator :** This operator provides the common elements in both lists.

- **Except operator :** Except the operator provides the element only from the first list. The new list does not contain common elements. Shown in example 4.



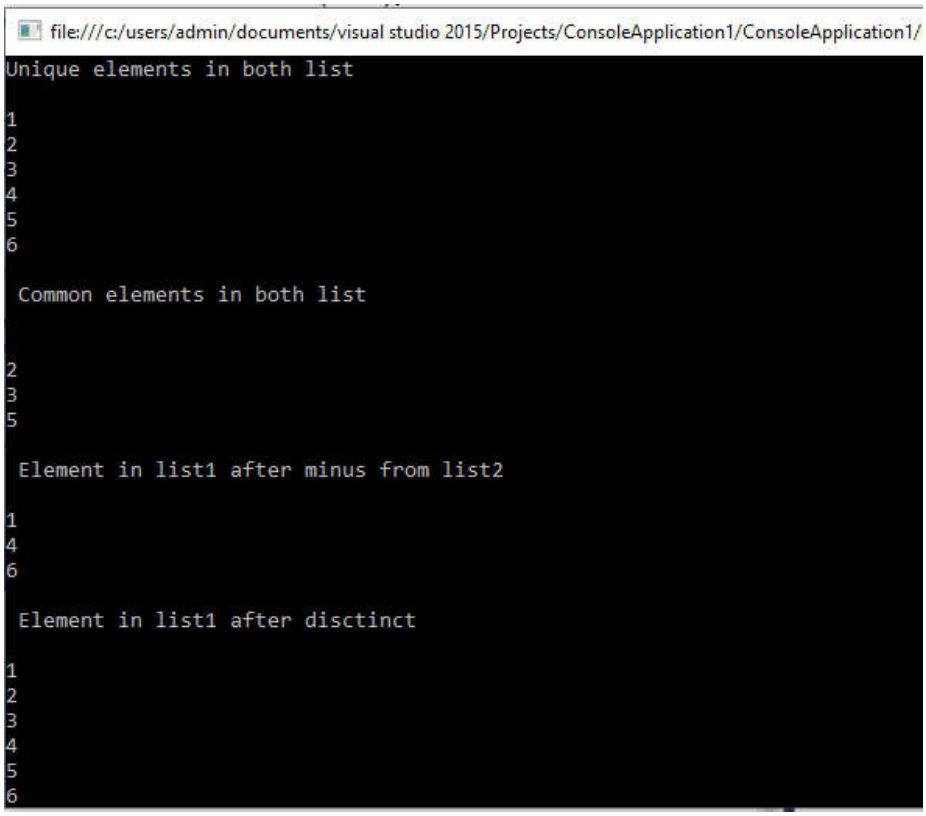
Example 4: LINQ with set operators.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
int[] List1 = {1,2,3,4,5,6 };
int[] List2 = {2,3,5};
Console.WriteLine("Unique elements in both list\n");
var result = List1.Union(List2);
foreach (int i in result)
{
Console.WriteLine(i);
}
Console.WriteLine("\n Common elements in both list\n\n");
var IntResult = List1.Intersect(List2);
foreach (int i in IntResult)
{
Console.WriteLine(i);
}
Console.WriteLine("\n Element in list1 after minus from list2 \n");
var ExceptResult = List1.Except(List2);
```

```
foreach (int i in ExceptResult)
{
Console.WriteLine(i);
}
Console.WriteLine("\n Element in list1 after disctinct \n");
var r = List1.Distinct();
foreach (int i in r)
{
Console.WriteLine(i);
}
Console.ReadLine();
}
}
}
```

Output:



LINQ Partition Operator

In LINQ, Partition Operators are used to partition the list/collections items into two parts and return one part of the list items. Here are the different types of partitioning operators available in LINQ

- **TAKE:** This operator is used to return the specified number of elements in the sequence.

- **TAKEWHILE:** This operator is used to return the elements in the sequence which satisfy the specific condition.
- **SKIP:** This operator is used to skip the specified number of elements in a sequence and return the remaining elements.
- **SKIPWHILE:** This operator is used to skip the elements in a sequence based on the condition, which is defined as true.

Quantifier Operators

- **"All"** operator checks whether all elements in the list satisfies the specified condition or not. It returns true if all the elements satisfy a condition otherwise it returns false.
- **"Any"** operator checks whether any element in the list satisfies the given condition or not. It returns true if any element satisfies the given condition. As it name indicates
- **"Contains"** operator checks whether a particular element exists in the collection or not. It also returns true or false. It returns true if any element is available in the given list.

LINQ Element Operators

In LINQ, element operators are used to return the first and last element of the list or single element from the collection or a specific element based on the index value from the collection. By using these element operators, we can get the list/collection of items at the specific position.

- **First**: It returns the first element in sequence or from the collection based on the condition.
- **FirstOrDefault:** It is the same as First, but it returns the default value in case when no element is found in the collection.
- **Last:** It returns the last element in sequence or the last element based on the matching criteria.
- **ElementAt:** It returns an element from the list based on the specific index position.
- **ElementAtOrDefault:** It is the same as ElementAt, but it returns the default value in case no element is present at the specified index of the collection.
- **Single:** It returns the single specific element from the collection.
- **SingleOrDefault:** It is the same as Single, but it returns the default value in case if no element is found in the collection.
- **DefaultifEmpty:** It returns the default value in case if the list or collection contains empty or null values.

16.4 SUMMARY

- LINQ functionality can be achieved by importing the System.Linq namespace in our application.
- Generally, the LINQ contains a set of extension methods which allows us to query the source of data objects directly in our code based on the requirement.
- In LINQ we have learnt LINQ introduction, syntax, min function, max function, sum function, count function, sorting operators and how it helps in timesaving.
- Ultimately, you have known about how language integrated queries can be used for several types of data sources.

16.5 REFERENCE FOR FURTHER READING

1. Beginning Visual C# 2012 Programming, Karli Watson, Jacob Vibe Hammer, Jon D. Reid, Morgan Skinner, Daniel Kemper, Christian Nagel,ISBN: 978-1-118-31441-8,Wrox Publication
2. Professional C# 2008,Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner,ISBN: 978-1-118-64321-1 ,Wrox Publication
3. Murach's C# 2015,Anne Boehm and Joel Murach, ISBN 978-1-890774-94-3,Murrach Books

16.6 UNIT END EXERCISES

1. Explain LINQ with examples.
2. Explain different types of LINQ operators.
3. What are the different ways to implement LINQ?
4. What are the steps for creating LINQ to objects?
5. What are the steps for creating LINQ to XML?
6. What are the Advantages and Disadvantages of LINQ?

❄❄❄❄❄❄❄

17

ASP.NET AJAX

Unit Structure :

17.0 OBJECTIVE

- To understand the use ASP.NET AJAX toolkit.
- To use several features of AJAX in ASP.NET.
- To learn how to use it to create the next generation of highly interactive, dynamic web pages.

17.1 INTRODUCTION

- ASP.NET AJAX consists of two key parts: a client-side portion and a server-side portion.
    - The **client-side** portion is a set of JavaScript libraries.
    - non-ASP.NET developers can use them in their own web pages.
    - The client libraries don't expose much in the way of features. Rather, they establish a basic foundation you can use to develop ASP.NET AJAX pages.
    - This foundation extends the JavaScript language to fill in a few of its gaps, and provides some basic infrastructure.
    - The **server-side** portion of ASP.NET AJAX works at a higher level.
    - It includes controls and components that use the client-side JavaScript libraries. For example, a web form that contains the DragPanel component gives users the ability to drag a panel around in the browser window.

- JavaScript uses the client-side ASP.NET AJAX libraries. However, the DragPanel renders all the JavaScript code it needs.

- ASP.NET AJAX is the start of a new direction in ASP.NET development. Before going any further, it's worth getting an overview of all the features that ASP.NET AJAX provides.

- **List of analysis:**

  - **JavaScript language extensions:** These extensions make JavaScript work a little more like a modern object-oriented language, with support for namespaces, inheritance, interfaces, enumerations, and reflection.

  - **Remote method calls:** ASP.NET AJAX pages can call web services that you design. This feature allows users to get information from the server without performing a full-page postback.

  - **ASP.NET services:** This feature allows you to call the server to use one of two ASP.NET services one that uses forms authentication information and one that gets data from the current user profile.

  - **Partial page refreshes:** The new UpdatePanel control gives you a way to define a portion of a page that will be updated without requiring a full-page postback.

  - **Prebuilt controls:** The popular ASP.NET AJAX Control Toolkit is stocked with more than 30 controls and control extenders that use ASP.NET AJAX to great effect. They allow you to make controls collapse and expand, add dynamic animations, and support autocompletion and drag-and-drop. And once again, these classes handle the low-level JavaScripts.

ASP.NET AJAX on the Client: The Script Libraries

- The client-side portion of ASP.NET AJAX relies on a small collection of JavaScript files. There are two ways to deploy the ASP.NET AJAX script files. If you build an ASP.NET application, they're available through the System.Web.Extensions.dll assembly and served out on demand.

- In ASP.NET, you won't find individual JavaScript files for the client libraries. Instead, the client libraries are embedded in the System.Web.Extensions.dll assembly and served up as a script resource. Script resources allow you to map a URL to a resource that's embedded in an assembly.

- For example, example 2 shows a sample script block that extracts the ASP.NET AJAX script library:

Example 2:

```
<script
    src="/YourWebSite/ScriptResource.axd?d=RUSU1mIv69CJ9
    H5JUAOSw8L4674

    LfxGOQg6Nw7HtNHheB3bMiw7Ov16bX1KPG6N1oTYEi6
    5ggRoIP1-hWapSttV3udoNXGrk095YGEzuX0M1&am

    p;t=633127440334523405" type="text/javascript">

    </script>
```

ASP.NET includes a script resource handler that responds to these requests. It examines the passed- in query string argument and returns the requested script file.

ASP.NET AJAX on the Server: The ScriptManager

- To type long URLs there is no need to script resources on every page that requires ASP.NET AJAX. The solution is to use an ASP.NET control called the ScriptManager.
- The ScriptManager is the brains of the server-side ASP.NET AJAX model. It's a web control that doesn't have any visual appearance on the page.
- It performs a key task: it renders the links to the ASP.NET AJAX JavaScript libraries.
- To add the ScriptManager to a page, you can drag it from the AJAX Extensions tab of the Toolbox.

ScriptManager is declared in the .aspx file:

```
<asp:ScriptManager ID="ScriptManager1"
    runat="server"></asp:ScriptManager>
```

- Each page that uses ASP.NET AJAX features requires an instance of the ScriptManager.
- Allows only one ScriptManager on a page. Along with rendering the links for the ASP.NET AJAX client libraries, the ScriptManager also performs several other important tasks.
- It can render references to other script files, create proxies that enable it to call web services asynchronously from the browser, and manage the way UpdatePanel controls refresh their content.

17.2 SCRIPTMANAGER

- Manages ASP.NET Ajax script libraries and script files, partial-page rendering, and client proxy class generation for Web and application services.

Namespace:

System.Web.UI

Assembly:

System.Web.Extensions.dll

[System.Drawing.ToolboxBitmap(typeof(EmbeddedResourceFinder), "System.Web.Resources.ScriptManager.bmp")]

public class ScriptManager : System.Web.UI.Control, System.Web.UI.IPostBackDataHandler, System.Web.UI.IPostBackEventHandler

The following examples show different scenarios for using the ScriptManager control.

Enabling Partial-Page Updates

- Example 3: a Calendar and a DropDownList control are inside an UpdatePanel control. By default, the value of the UpdateMode property is Always, and the value of the ChildrenAsTriggers property is true. Therefore, child controls of the panel cause an asynchronous postback.

Example 3:

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
   void DropDownSelection_Change(Object sender, EventArgs e)
   {
      Calendar1.DayStyle.BackColor =
         System.Drawing.Color.FromName(ColorList.SelectedItem.Value);
   }
   protected void Calendar1_SelectionChanged(object sender, EventArgs e)
   {
      SelectedDate.Text =
         Calendar1.SelectedDate.ToString();
   }
```

```
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
   <title>UpdatePanel Example</title>
</head>
<body>
   <form id="form1" runat="server">
      <div>
         <asp:ScriptManager ID="ScriptManager1"
                     runat="server" />
         <asp:UpdatePanel ID="UpdatePanel1"
                     runat="server">
            <ContentTemplate>
               <asp:Calendar ID="Calendar1"
                           ShowTitle="True"

OnSelectionChanged="Calendar1_SelectionChanged"
                           runat="server" />
               <div>
                  Background:
                  <br />
                  <asp:DropDownList ID="ColorList"
                              AutoPostBack="True"

OnSelectedIndexChanged="DropDownSelection_Change"
                              runat="server">
                     <asp:ListItem Selected="True" Value="White">
                     White </asp:ListItem>
                     <asp:ListItem Value="Silver">
                     Silver </asp:ListItem>
                     <asp:ListItem Value="DarkGray">
                     Dark Gray </asp:ListItem>
                     <asp:ListItem Value="Khaki">
                     Khaki </asp:ListItem>
                     <asp:ListItem Value="DarkKhaki"> D
                     ark Khaki </asp:ListItem>
                  </asp:DropDownList>
               </div>
               <br />
               Selected date:
               <asp:Label ID="SelectedDate"
```

```
                    runat="server">None.</asp:Label>
            </ContentTemplate>
        </asp:UpdatePanel>
        <br />
    </div>
  </form>
</body>
</html>
```

Handling Partial-Page Update Errors and Registering Script

- This provides custom error handling during partial-page updates.
- By default, when an error occurs during partial-page updates, a JavaScript message box is displayed.
- This example shows how to use custom error handling by providing a handler for the AsyncPostBackError event, and by setting the AsyncPostBackErrorMessage property in the event handler.
- We can set the AllowCustomErrorsRedirect property to specify how the custom errors section of the Web.config file is used when an error occurs during partial-page updates.
- In this example 4, the default value of the AllowCustomErrorsRedirect property is used. This means that if the Web.config file contains a customErrors element, that element determines how errors are displayed.

Example 4:

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<script runat="server">

  protected void Button1_Click(object sender, EventArgs e)

  {

    try

    {

      int a = Int32.Parse(TextBox1.Text);

      int b = Int32.Parse(TextBox2.Text);

      int res = a / b;
```

```
            Label1.Text = res.ToString();
        }
        catch (Exception ex)
        {
            if (TextBox1.Text.Length > 0 && TextBox2.Text.Length > 0)
            {
                ex.Data["ExtraInfo"] = " You can't divide " +
                    TextBox1.Text + " by " + TextBox2.Text + ".";
            }
            throw ex;
        }
    }
    protected void ScriptManager1_AsyncPostBackError(object sender,
AsyncPostBackErrorEventArgs e)
    {
        if (e.Exception.Data["ExtraInfo"] != null)
        {
            ScriptManager1.AsyncPostBackErrorMessage =
                e.Exception.Message +
                e.Exception.Data["ExtraInfo"].ToString();
        }
        else
        {
            ScriptManager1.AsyncPostBackErrorMessage =
                "An unspecified error occurred.";
        }
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>UpdatePanel Error Handling Example</title>
    <style type="text/css">
    #UpdatePanel1 {
```

```
    width: 200px; height: 50px;
    border: solid 1px gray;
  }
  #AlertDiv{
  left: 40%; top: 40%;
  position: absolute; width: 200px;
  padding: 12px;
  border: #000000 1px solid;
  background-color: white;
  text-align: left;
  visibility: hidden;
  z-index: 99;
  }
  #AlertButtons{
  position: absolute; right: 5%; bottom: 5%;
  }
  </style>
</head>
<body id="bodytag">
  <form id="form1" runat="server">
    <div>
      <asp:ScriptManager ID="ScriptManager1"
      OnAsyncPostBackError="ScriptManager1_AsyncPostBackError"
runat="server" >
      <Scripts>
      <asp:ScriptReference Path="ErrorHandling.js" />
      </Scripts>
      </asp:ScriptManager>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
          <asp:TextBox ID="TextBox1" runat="server"
Width="39px"></asp:TextBox>
          /
```

```
            <asp:TextBox ID="TextBox2" runat="server"
Width="39px"></asp:TextBox>
            =
            <asp:Label ID="Label1" runat="server"></asp:Label><br />
            <asp:Button ID="Button1" runat="server"
OnClick="Button1_Click" Text="calculate" />
          </ContentTemplate>
        </asp:UpdatePanel>
        <div id="AlertDiv">
          <div id="AlertMessage">
          </div>
          <br />
          <div id="AlertButtons">
            <input id="OKButton" type="button" value="OK"
runat="server" onclick="ClearErrorState()" />
          </div>
        </div>
      </div>
  </form>
</body>
</html>
```

Globalizing the Date and Time That Are Displayed in the Browser

- The following example 5 shows how to set the EnableScriptGlobalization property so that the client script can display a culture-specific date and time in the browser.
- In the code, the Culture attribute of the @ Page directive is set to auto. As a result, the first language that is specified in the current browser settings determines the culture and UI culture for the page.

Example 5:

```
<%@ Page Language="C#" Culture="auto" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Globalization Example</title>
```

```
</head>
<body>
   <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1"
EnableScriptGlobalization="true" runat="server">
      </asp:ScriptManager>
      <script type="text/javascript">
      function pageLoad() {
        Sys.UI.DomEvent.addHandler($get("Button1"), "click",
formatDate);
      }
      function formatDate() {
        var d = new Date();
        try {
          $get('Label1').innerHTML = d.localeFormat("dddd, dd MMMM
yyyy HH:mm:ss");
        }
        catch(e) {
          alert("Error:" + e.message);
        }
      }
      </script>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server"
ChildrenAsTriggers="False" UpdateMode="Conditional">
      <ContentTemplate>
      <asp:Panel ID="Panel1" runat="server" GroupingText="Update
Panel">
      <asp:Button ID="Button1" runat="server" Text="Display Date" />
      <br />
      <asp:Label ID="Label1" runat="server"></asp:Label>
      </asp:Panel>
      </ContentTemplate>
      </asp:UpdatePanel>
   </form>
</body>
</html>
```

17.3 PARTIAL REFRESHES

- Perhaps the most visible feature of the ASP.NET AJAX Extensions is the ability to do a partial or incremental page update without doing a full postback to the server, with no code changes and minimal markup changes.
- The advantages are extensive: the state of your multimedia (such as Adobe Flash or Windows Media) is unchanged, bandwidth costs are

reduced, and the client does not experience the flicker usually associated with a postback.

- The ability to integrate partial page rendering is integrated into ASP.NET with minimum changes into your project.
- One of the most fascinating controls in the ASP.NET AJAX framework is the UpdatePanel. This new control replaces the need for a page to refresh during a postback. Only portions of a page, designated by the UpdatePanel, are updated.
- This technique is known as partial-page rendering and can be highly effective in improving the user experience.

17.4 PROGRESS NOTIFICATION

1. Showing a Simulated Progress Bar

- When adding the UpdateProgress control to a page, we get the ability to specify some content that will appear as soon as an asynchronous request is started and disappear as soon as the request is finished.
- This content can include a fixed message, but many people prefer to use an animated GIF, because it more clearly suggests that the page is still at work.
- The top figure shows the page as it first appears, with a straightforward UpdatePanel control containing a button. When the button is clicked, the asynchronous callback process begins. At this point, the contents of the UpdateProgress control appear underneath.
- The markup for this page defines an UpdatePanel followed by an UpdateProgress example 6 shown below:

Example 6:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <div style="background-color:#FFFFE0;padding: 20px">
     <asp:Label ID="lblTime" runat="server" Font-
Bold="True"></asp:Label>
      <br /><br />
     <asp:Button ID="cmdRefreshTime" runat="server"
      Text="Start the Refresh Process" />
    </div>
  </ContentTemplate>
</asp:UpdatePanel>
<br />
```

```
<asp:UpdateProgress ID="updateProgress1" runat="server">
  <ProgressTemplate>
    <div style="font-size: xx-small">
      Contacting Server ... <img src="wait.gif" alt="Waiting..." />
    </div>
  </ProgressTemplate>
</asp:UpdateProgress>
```

- Depending on the layout we want, we can place our UpdateProgress control somewhere inside our UpdatePanel control.
- The UpdateProgress control only shows its content while the asynchronous callback is under way, it only makes sense to use it with an operation that takes time. Otherwise, the UpdateProgress will only show its ProgressTemplate for a few fractions of a second.
- To simulate a slow process, add a line to delay your code 10 seconds, as example 7 shown below:

  Example 7:

```
Protected Sub cmdRefreshTime_Click(ByVal sender As Object, _
  ByVal e As EventArgs) Handles cmdRefreshTime.Click

System.Threading.Thread.Sleep(TimeSpan.FromSeconds(10))
    lblTime.Text = DateTime.Now.ToLongTimeString()
End Sub
```

2. Cancellation

- The UpdateProgress control supports one other detail: a cancel button. When the user clicks a cancel button, the asynchronous callback will be cancelled immediately, the UpdateProgress content will disappear, and the page will revert to its original state.
- Adding a cancel button is a two-step process. First you need to add a fairly intimidating block of JavaScript code, which you can copy verbatim. You should place this code at the end of your page, after all your content but just before the </body> end tag. Here's the example 8 shows the need, in its rightful place:

  Example 8:

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="WaitIndicator.aspx.vb"
 Inherits="WaitIndicator" %>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
   ...
</head>
<body>
 <form ID="form1" runat="server">
   ...
 </form>
 <script type="text/javascript">
   var prm = Sys.WebForms.PageRequestManager.getInstance();
   prm.add_initializeRequest(InitializeRequest);
   function InitializeRequest(sender, args)
   {
      if (prm.get_isInAsyncPostBack())
      {
         args.set_cancel(true);
      }
   }
   function AbortPostBack()
   {
      if (prm.get_isInAsyncPostBack()) {
       prm.abortPostBack();
      }
   }
   </script>
</body>
</html>
```

- Once you've added this code, you can use JavaScript code to call its AbortPostBack() function at any time and cancel the callback.
- The easiest way to do this is to connect a JavaScript event to the AbortPostBack() function using a JavaScript event attribute. Add a JavaScript event attribute to virtually any HTML element. For example, you can deal with client-side clicks using the onclick attribute. Here's a basic HTML button that uses this technique to connect itself to the AbortPostBack() function:

  <input ID="cmdCancel" onclick="AbortPostBack()" type="button" value="Cancel" />

- If you click this Cancel button, the client-side AbortPostBack() function will be triggered and the callback will be cancelled immediately. Typically, this button in the ProgressTemplate of the UpdateProgress control, as shown in Figure 1.

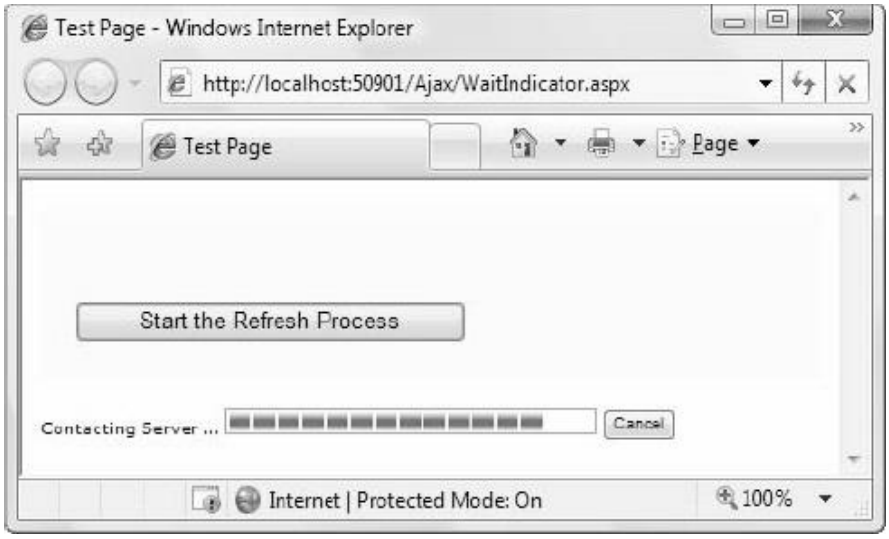


Figure 1 ProgressTemplate

17.5 TIMED REFRESHES

- ASP.NET AJAX includes a Timer control that can help you implement a page that includes a stock ticker, and want to refresh this ticker periodically (say, every 5 minutes) to ensure it doesn't become drastically outdated.
- The Timer control is refreshingly straightforward. You simply add it to a page and set its Interval property to the maximum number of milliseconds that should elapse before an update.
- For example 9 & 10, if you set Interval to 60000, the timer will force a postback after one minute elapses.

  <asp:Timer ID="Timer1" runat="server" Interval="60000" />
- If the Timer is in an UpdatePanel, it will trigger an asynchronous postback. If it's not, and it's not linked to an UpdatePanel with a trigger, the Timer will trigger an ordinary full-page postback.
- The timer raises a server-side Tick event, which you can handle to update your page.
- The timer is particularly well suited to pages that use partial rendering. That's because a refresh in a partially rendered page might just need to change a single portion of the page.
- To use the timer with partial rendering, wrap the updateable portions of the page in UpdatePanel controls with the UpdateMode set to Conditional, and add a trigger that forces an update whenever the timer fires:

Example 9:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
<ContentTemplate>
...
</ContentTemplate>
<Triggers>
<asp:AsyncPostBackTrigger ControlID="Timer1" EventName="Tick" />
</Triggers>
</asp:UpdatePanel>
<asp:Timer ID="Timer1" runat="server" Interval="60000" OnTick="Timer1_Tick" />
```

- To stop the timer, simply need to set the Enabled property to false in server-side code. For example, following code show how to disable the timer after ten updates:

Example 10:

```
protected void Timer1_Tick(object sender, EventArgs e)
{
// Update the tick count and store it in view state.
int tickCount = 0;
if (ViewState["TickCount"] != null)
{
tickCount = (int)ViewState["TickCount"];
}
tickCount++;
ViewState["TickCount"] = tickCount;
// Decide whether to disable the timer.
if (tickCount > 10)
{
Timer1.Enabled = false;
}
}
```

17.6 SUMMARY

- The most exciting feature of ASP.NET AJAX is that it isn't just another JavaScript library or a simple .NET component that simplifies callbacks.
- You can write your own JavaScript code that calls server-side functionality.
- We can keep using ordinary ASP.NET server controls but extend them with ASP.NET AJAX-fortified ingredients such as the UpdatePanel, or use the snazzy controls and control extenders that are included with the ASP.NET AJAX Control Toolkit.
- We can create your own client-side components, controls, and behaviors, and use them independently or in conjunction with a custom ASP.NET server control.

17.7 REFERENCE FOR FURTHER READING

1. Beginning ASP.NET 4.5 in C#, Matthew MacDonald, Apress(2012)
2. The Complete Reference ASP .NET, MacDonald, Tata McGraw Hill
3. Beginning ASP.NET 4 in C# and VB Imar Spanajaars, WROX

17.8 UNIT END EXERCISES

1. What is the use of ScriptManager in ASP.NET AJAX?
2. Explain the Partial Refreshes with an example.
3. Explain Progress Notification with the help of an example?
4. Write a short note on Timed Refreshes?

❄❄❄❄❄❄❄