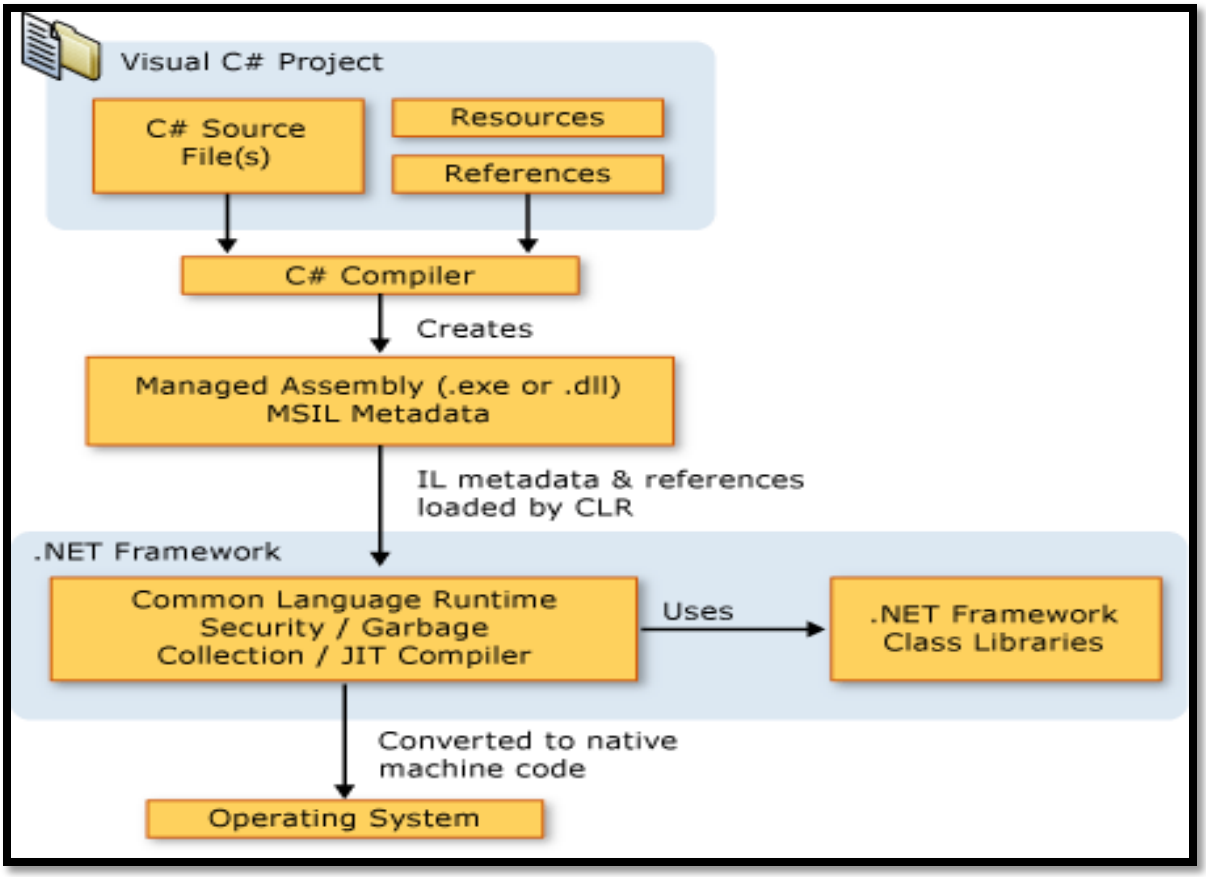
CHAPTER 1: C# - BASIC SYNTAX

Contents

1.0 Review of .NET Frameworks



The **.NET Framework** is a **development platform** for building apps for web, Windows, Windows Phone, Windows Server, and Microsoft Azure. The .NET Framework is a **managed execution environment** for Windows that provides a variety of services to its running apps. It consists of **two major components**: **the common language runtime (CLR)**, which is the execution engine that handles running apps, and the **.NET Framework Class Library**, which provides a library of tested, reusable code that developers can call from their own apps. The services that the .NET Framework provides to running apps include the following:

1) **Memory management.** In .NET Framework apps, the CLR allocates and releases memory and for handling object lifetimes on behalf of the app.
2) **A common type system.** In the .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target the .NET Framework.
3) **An extensive class library.** Instead of having to write vast amounts of code to handle common low-level programming operations, programmers use a readily accessible library of types and their members from the .NET Framework Class Library.
4) **Development frameworks and technologies.** The .NET Framework includes libraries for specific areas of app development, such as ASP.NET for web apps, ADO.NET for data

access, Windows Communication Foundation for service-oriented apps, and Windows Presentation Foundation for Windows desktop apps.

5) **Language interoperability.** Language compilers that target the .NET Framework emit an intermediate code named Common Intermediate Language (CIL), which, in turn, is compiled at runtime by the common language runtime. With this feature, routines written in one language are accessible to other languages, and programmers focus on creating apps in their preferred languages.
6) **Version compatibility.** With rare exceptions, apps that are developed by using a particular version of the .NET Framework run without modification on a later version.
7) **Side-by-side execution.** The .NET Framework helps resolve version conflicts by allowing multiple versions of the common language runtime to exist on the same computer and an app can run on the version of the .NET Framework with which it was built.
8) **Multitargeting.** By targeting .NET Standard, developers create class libraries that work on multiple .NET Framework platforms supported by that version of the standard.

1.1 Introduction to C#

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, etc.

The syntax of C# is 70% Java, 10% C++, 5% Visual Basic, 15% new.C# provides powerful features such as nullable value types, enumerations, delegates, lambda expressions and direct memory access. C# supports generic methods and types, collection classes having enumerators and iterators that are simple to use by client code.

C# supports the object-oriented programming concepts of encapsulation, inheritance, and polymorphism. All variables and methods are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. It supports method overloading and method overriding. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, it has several innovative language constructs, including the following:Encapsulated method signatures called delegates, which enable type-safe event notifications.Properties, which serve as acessors for private member variables.Attributes, which provide declarative metadata about types at run time.Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.

1.2 Variables and Expressions

1.2.1 Identifier

An **identifier**, in C#, is the user-defined name of a program element. It can be a namespace, class, method, variable or interface. Identifiers are symbols used to uniquely identify a programelement in the code. They are also used to refer to types, constants, macros and parameters. The identifier can begin with either a letter (uppercase or lowercase) or a underscore ('-') or the symbol '@'. The succeeding characters can be any letter or digit or '-'. May contain Unicode escape sequences (e.g. \u03c0 for p).

1.2.2 Variable

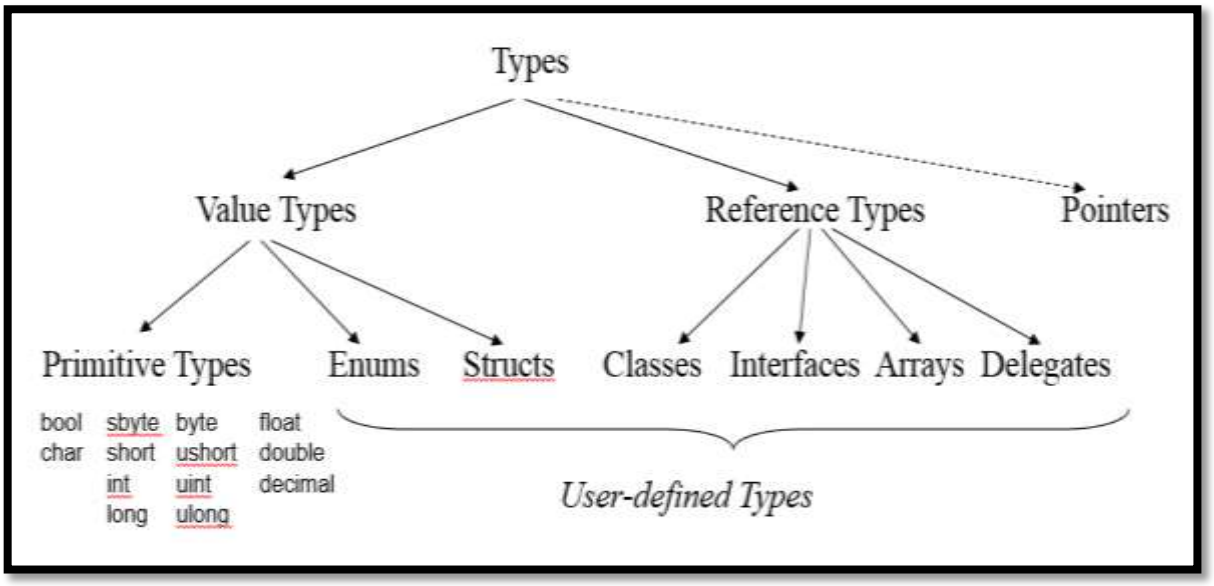
The **variable** is a name given to a data value. A variable holds the value of specific type e.g string, int, etc. A variable can be declared and initialized later or declared & initialized at the same time. The value of a variable can be changed at any time throughout the program as long as it is accessible.Examples: someName, sum_of3, _10percent, @while, \u03c0.

1.2.3 Keyword

Keywords are predefined, reserved identifiers that have special meanings to the compiler. They cannot be used as identifiers in your program unless they include @ as a prefix. There are 77 keywords. Some of them are: is, base, checked, decimal, delegate, event,explicit,extern,fixed,foreach, implicitininternal, is,lock,object,override,params,readonly,ref,sealed,stackalloc, unchecked,unsafe,using.

1.2.4 Data Type

C# contains two general categories of built-in **data types**: **value types** and **reference types**.



All types are compatible with *object*
- can be assigned to variables of type *object*
-all operations of type *object* are applicable to them

Difference between Value Type and Reference Type

Variable of ...	**Value Types**	**Reference Types**
contains	value	reference
stored on	stack (or in an object)	heap
initialization	0, false, '\0'	null
assignment	copies the value	copies the reference
example	inti = 17; int j = i;	string s = "Hello"; string s1 = s;

1.2.5 Primitive Type

Primitive Types are non-numeric and numeric. The non-numeric are bool and char. The former represents the values true/false. The latter is 16-bit quantity to hold a Unicode character, which defines the character set for all languages of the world.

	long form	**range**
sbyte	System.SByte	-128 .. 127
byte	System.Byte	0 .. 255
short	System.Int16	-32768 .. 32767
ushort	System.UInt16	0 .. 65535
int	System.Int32	-2^{31} .. $2^{31}-1$
uint	System.UInt32	0 .. $2^{32}-1$
long	System.Int64	-2^{63} .. $2^{63}-1$
ulong	System.UInt64	0 .. $2^{64}-1$
float	System.Single	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	true, false
char	System.Char	Unicode character

The numeric type are integral types to represent integral decimal, octal and hexadecimal values; and the floating types to represent floating point values. The decimal is a floating type for financial calculations having higher precision. It provides 28-29 significant digits.

1.2.6 Literals

The **Literals** are fixed values in human readable form.

The **bool literals** are true or false.

The **char literals** are enclosed by single quotation marks. E.g. ‘a’, ‘3’, ‘&’. They can also be represented by their Unicode hexadecimal value. E.g. \0x0041 is ‘a’; \0x0391 is Greek letter ‘α’.

The **string literal** is enclosed within double quotes. E.g. “C”, “Hello World”, “1234”.

Integer literals are specified as number. E.g. 10, -100 are decimal integers. 045 is an octal integer. 0xCD87 and 0x ffb6 are hexadecimal integers. Examples of floating point literals are 1.23, 4.5e20, 7.78E-12.

The **decimal literal** is specified by appending a ‘m’ or ‘M’ at the end. E.g. 5.3445m, 7.123345M.

When a char literal begins with the symbol ‘\’ (called as escape sequence then the letter which follows it has special meaning. E.g. ‘\n’ indicates a new line; ‘\t’ is a horizontal tab; ‘\\’ is a backslash; ‘\”’ is a double quote.

Integer literals, the type of the integer literal is the smallest integer type that will holdit, beginning with int. Thus, an integer literal is either of type int, uint, long, or ulong,depending upon its value. Floating-point literals are of type double.

If you do not want the C#’s default type for a literal, you can explicitly specify its type by including a suffix. To specify a long literal, append an l or an L. E.g. 25L is a long.To specify an unsigned integer value, append a u or U. E.g. 513U is a uint. To specify an unsigned, long integer, use ul or UL. E.g.1234789UL is of type ulong. To specify a float literal, append an F or f to the constant. E.g. 10.19F is of type float.

1.2.7 Operators

C# has a rich set of **operators**which allows the programmer to construct varied types of expressions. Most of them are similar to those found in C++ and other modern languages. They can be categorized in various groups: **Arithmetic, relational, logical, bitwise, shift, assignment, compound assignment,** etc. There are other operators which handle specialized

situations, like indexing an array, accessing the members of class, etc.**Operator precedence** is a set of rules which defines how an expression is evaluated.But if both the operators have same precedence, then the expression is evaluated based on the **associativity of operator** (left to right or right to left).

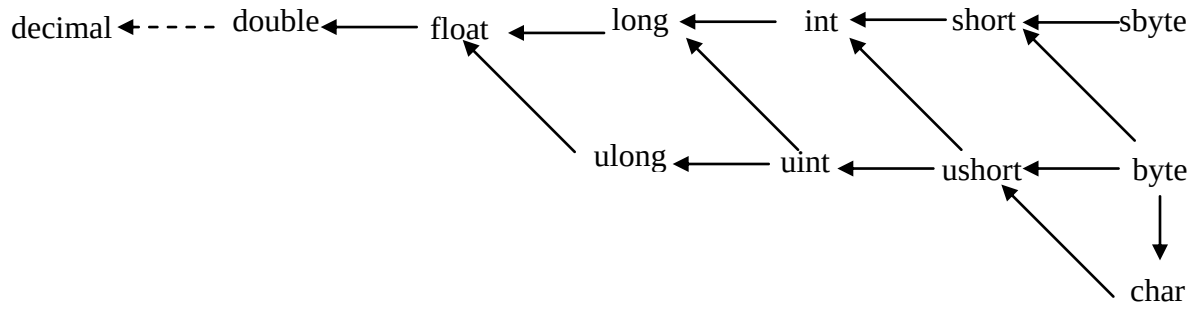
C# Operator Precedence (Highest to lowest)	
Category	**Operators**
Postfix Increment and Decrement	**++, --**
Prefix Increment, Decrement & Unary	**++, --, +, -, !, ~**
Multiplicative	***, /, %**
Additive	**+, -**
Shift	**<<, >>**
Relational	**<, <=, >, >=**
Equality	**==, !=**
Bitwise AND	**&**
Bitwise XOR	^
Bitwise OR	|
Logical AND	**&&**
Logical OR	||
Ternary	**? :**
Assignment	**=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=**

C# Associativity of operators		
Category	**Operators**	**Associativity**
Postfix Increment and Decrement	**++, --**	Left to Right
Prefix Increment, Decrement & Unary	**++, --, +, -, !, ~**	Right to Left
Multiplicative	***, /, %**	Left to Right
Additive	**+, -**	Left to Right
Shift	**<<, >>**	Left to Right
Relational	**<, <=, >, >=**	Left to Right
Equality	**==, !=**	Left to Right
Bitwise AND	**&**	Left to Right
Bitwise XOR	^	Left to Right
Bitwise OR	|	Left to Right
Logical AND	**&&**	Left to Right
Logical OR	||	Left to Right
Ternary	**? :**	Right to Left
Assignment	**=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=**	Right to Left

1.2.8 Type Casting

A common task in programming is assignment of one type of variable to another. These conversions can be automatic (implicit) or explicit, which is called as **casting**. The first conversions will occur when the two types are compatible and the range of destination type is wider than the source type. If the second condition is not fulfilled, then we have to perform a cast. The general syntax for casting is :(target-type) expression. E.g. if a, b are of the type double and val is of type int then we can write val = (int) (a / b).

The automatic conversions are given below. For example, if the expression contains int type and long type, then the int type is automatically promoted to the long type, and so forth.



1.2.9 Boxing and Unboxing

Simple types and other structs inherit from class ValueType in namespace System. ClassValueType inherits from class object. Thus, any simple-type value can be assigned to an object variable; this is referred to as a **boxing conversion** and enables simple types to be used anywhere objects are expected. In a boxing conversion, the simple-type value is copied into an object so that the simple-type value can bemanipulated as an object. Boxing conversions can be performed either explicitly or implicitly as shown in the following statements:

```
inti = 50;                      // create an int value

object object1 = ( object ) i;  // explicitly box the int value

object object2 = i;             // implicitly box the int value
```

An **unboxing conversion** can be used to explicitly convert an object reference to a simple value, as shown in the following statement:

```
int int1 = ( int ) object1;     // explicitly unbox the int value
```

Explicitly attempting to unbox an object reference that does not refer to the correct simple value type causes an error.

1.2.10 Arrays

An **array** is a group of variables (called elements) containing values that all have the same type. Arrays are reference types. An array is actually a reference to an array object. The elements of an array can be either value types or reference types, including other arrays. To refer to a particular element in an array, we specify the name of the reference to the array and the position number of the element in the array, which is known as the element's index. The first element in every array has index zero. Array names follow the same conventions as other variable names.An index must be a nonnegative integer and can be an expression.

Arrays in C# can be single dimension, or multi dimension. Two dimensional array are of two types: rectangular or jagged. In a rectangular array, the number of columns in each row is same, whereas in a jagged array the number of columns in all rows is not equal.

Creation of arrays

One-dimensional arrays

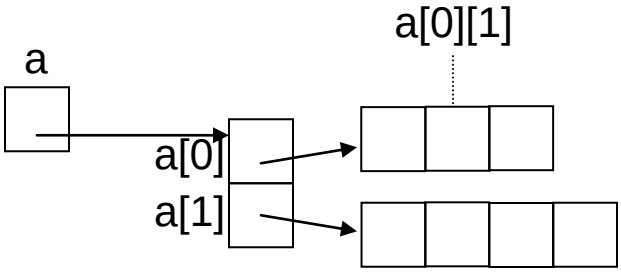
```
int[] a = new int[3];
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};
SomeClass[] d = new SomeClass[10];       // array of references
SomeStruct[] e = new SomeStruct[10];     // array of values (directly in the array)
```

To access the second element of array a, we can write a[1].

Multidimensional arrays (jagged)

```
int[][] a = new int[2][];       // array of references to other arrays
a[0] = new int[] {1, 2, 3};     // cannot be initialized directly
a[1] = new int[] {4, 5, 6};
```

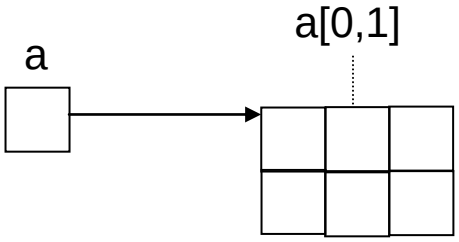
To access the second element of array a (first row and second column, we can write a[0][1].



Multidimensional arrays (rectangular)

```
int[,] a = new int[2, 3];           // block matrix
int[,] b = {{1, 2, 3}, {4, 5, 6}};  // can be initialized directly
```

To access the second element of array a (first row and second column), we can write a[0,1].



Each array has associated with it a **Length** property that contains the number of elements that an array can hold. Thus, each array provides a means by which its length can be determined.

1.2.11 Expressions

An **expression** is a sequence of one or more operands and zero or more operators that can be evaluated to a single value, object, method, or namespace. Expressions can consist of a literal value, a method invocation, an operator and its operands, or a simple name. Simple names can be the name of a variable, type member, method parameter, namespace or type.

1.2.12 Statements

The actions that a program takes are expressed in **statements**. Common actions include declaring variables, assigning values, calling methods, looping through collections, and branching to one or another block of code, depending on a given condition.A statement can consist of a single line of code that ends in a semicolon, or a series of single-line statements in a block. A statement block is enclosed in {} brackets and can contain nested blocks.

1.2.13 Comments

A **comment** describes or explains the operation of the program to anyone who is reading its source code. The contents of a comment are ignored by the compiler. Three types of comments are:

Single-line comments

```
// This is a comment till the end of this line
```

Delimited comments

```
/* This is a comment which
can span several lines*/
```

This must not be nested.

Documentation comments

```
/// This is a documentation comment
```

1.3Flow Control Structures

The order in which statements are executed in a program is called the **flow of control** or flow of execution. The flow of control may vary every time that a program is run, depending on how the program reacts to input that it receives at run time. The syntax and working of the control structures are similar to those in C++.

There are three types of **control structures**—sequence, selection and repetition. The sequence structure is built into C#. Unlessdirected otherwise, the computer executes C# statements one after the other in the order in which they're written.

1.3.1 Selection Statements

C# has three types of selection statements: the if statement, the if…else statement and the switch statement.

The **if statement** is called a single-selection statement becauseit selects or ignores a single action.

```
if(condition) statement;
```

The **if…else statement** is called a double-selection statement because it selects between two different actions (or groups of actions).

```
if(condition) statement;
else statement;
```

The **switch statement** is called a multiple-selection statement because it selects among many different actions (or groups ofactions).

```
switch(expression) {
        case constant1:
                statement sequence
                break;
        case constant2:
                statement sequence
        break;
        .
        .
        default:
                statement sequence
```

```
                    break;
}
```

1.3.2 Repetition Statements

C# provides four repetition statements: the while, do…while, for and foreach statements.

The **while, for and foreach statements** perform the actions intheir bodies zero or more times.

```
for(initialization; condition; iteration) statement;

while(condition) statement;

foreach(type loopvar in collection) statement;
```

The **do…while** statement performs the actions in its body one ormore times.

```
do {
        statements;
} while(condition);
```

To include several statements in the body of an if (or the body of an else for an if…else statement), enclose the statements in braces ({ and }).A set of statements contained within a pair of braces is called a**block**. A block can be placed anywhere in an app that a singlestatement can be placed.

The statement or statement block of the body of each control structure is executed when the ‘condition’ expression evaluates to ‘true’. All these control structures can be nested within itself or others.

1.3.3Break and Continue Statements

C# provides statements break and continue to alter the flow of control.

The **break statement** causes immediate exit from a while, for, do…while, switch or foreach statement. Execution typically continues with the first statement after the control statement.

The **continue statement**, when executed in a while, for, do…whileor foreach, skips theremaining statements in the loop body andproceeds with the next iteration of the loop.

1.4Functions

Methods (called **functions** or **procedures** in other programminglanguages) allow you to modularize an app by separating its tasksinto self-contained units. The actual statements in the methodbodies are written only once,can be reused from several locationsin an app and are hidden from other methods.

General syntax of a method:

```
access ret-type name(parameter-list) {
        // body of method
}
```

The **'access'** is an access modifier that governs what other parts of your program can call the method. The access modifier is optional. If not present, then the method is private to the class in which it is declared.

The **'ret-type'** specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void.

The name of the method is specified by **'name'**. This can be any legal identifier other than those that would cause conflicts within the current declaration space.

The **'parameter-list'** is a sequence of type and identifier pairs separated by commas. Parameters are variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

The **return** statements are used in the methods. There are two forms of return: one for use in void methods (those that do not return a value) and one for returning values. The former when used immediately exits the method in which it is called. The latter are used to return the result of some task.

1.5 Debugging and Error Handling

The errors your program will encounter can be classified in three categories: runtime, syntax, and logic errors.

A syntax error is due to a misuse of the C# language in your code. The built-in Code Editor of Microsoft Visual Studio makes it extremely easy to be aware of syntax errors as soon as they occur.

A logic error is called a bug. Debugging is the process of examining code to look for bugs or to identify problems. Debugging is the ability to monitor the behavior of a variable, a class, or its

members throughout a program. Microsoft Visual C# provides many features to perform debugging operations.

The **debugger** is the program you use to debug your code. The code or application that you are debugging is called the debuggee.Probably the most fundamental way of examining code is to read every word and every line, with your eyes, using your experience as a programmer. But is not applicable for code which covers many pages or many files. Microsoft Visual Basic provides various tools and windows that you use, one window or a combination of objects for debugging. One of the tools you can use is the Standard toolbar that is equipped with various debugging buttons.

One of the primary pieces of information you want to get is the value that a variable is holding. A window named **Locals** is used to show that value. Normally, when you start debugging, the Locals window shows automatically.

Just as done when reading code with your eyes, the most basic way to monitor code is to execute one line at a time and see the results displayed before your eyes. To support this operation, the debugger provides what is referred to as stepping into.

The **Step Into** feature is a good tool to monitor the behavior of variables inside a method. This also allows you to know if a method is behaving as expected. Once you have established that a method is alright, you may want to skip it. Instead of executing one line at a time, the debugger allows you to execute a whole method at a time or to execute the lines in some methods while skipping the others. To support this, you use a feature named **Step Over**.

When executing a program, you can specify a section or line where you want the execution to pause, for any reason you judge necessary. This approach is useful if you have checked code up to a certain point and it looked alright. If you are not sure about code starting at a certain point, this can be your starting point.

A **breakpoint** on a line is the code where you want the execution to suspend. You must explicitly specify that line by creating a breakpoint. You can as well create as many breakpoints as you want. You can also remove a breakpoint you don't need anymore.

You can combine the Step Into and/or the Step Over feature with breakpoints. That is, you can examine each code line after line until you get to a specific line. This allows you to monitor the values of variables and see their respective values up to a critical section. To do this, first create one or more breakpoints, then proceed with steps of your choice.

1.6 Example Programs

Program 1: A Simple program

```
using System;
namespace HelloWorld{
        class Program{
                static void Main(string[] args){
                        System.Console.WriteLine("Hello World");
                        System.Console.Read();
                }
        }
}
```

Program 2: A program which takes a number from the user and prints the word form of it. E.g. if the user enters 852, the program prints EIGHT FIVE TWO.

```
1       using System;
2
3       namespace ToWords{
4               class Program{
5                       static void Main(string[] args){
6                               string s = "";
7                               intnum;
8                               int digit;
9                               System.Console.WriteLine("Enter the number: ");
10                              num = Int32.Parse(System.Console.ReadLine());
11                              while (num != 0){
12                                      digit = num % 10;
13                                      switch (digit){
14                                              case 0: s = "ZERO " + s;
15                                                      break;
16                                              case 1: s = "ONE " + s;
17                                                      break;
18                                              case 2: s = "TWO " + s;
19                                                      break;
20                                              case 3: s = "THREE " + s;
21                                                      break;
22                                              case 4: s = "FOUR " + s;
23                                                      break;
24                                              case 5: s = "FIVE " + s;
25                                                      break;
```

```
26                                          case 6: s = "SIX " + s;
27                                                  break;
28                                          case 7: s = "SEVEN " + s;
28                                                  break;
30                                          case 8: s = "EIGHT " + s;
31                                                  break;
32                                          case 9: s = "NINE " + s;
33                                                  break;
34                                  }
35                                  num = num/10;
36                          }
37                          System.Console.WriteLine("The number in words is :\n" + s);
38                          System.Console.Read();
39                  }
40          }
41  }
```

Program 3:A program to demonstrate the creation of arrays

```
1using System;
2
3namespace ArrayDemo{
4   class Program{
5      static void Main(string[] args){
6         //Method 1
7         int[] inum = {0,1,2,3,4,5,6,7,8,9};
8
9         //Method 2
10          double[] dnum;
11         dnum = new [] {0.0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };
12
13          //Method 3
14          int[] isqr;
15          isqr = new int[10];
16
17          //Method 4
18          double[] dsqrt = new double[10];
19
20          for (inti = 0; i<inum.Length; ++i){
21             isqr[i] = inum[i] * inum[i];
22          }
23
```

```
24      for (inti = 0; i<inum.Length; ++i){
25         dsqrt[i] = Math.Sqrt(dnum[i]);
26      }
27
28      Console.WriteLine("\nDisplay using foreach\n");
29      foreach(int v in isqr){
30         Console.WriteLine("{0,7:D}",v);
31      }
32
33      Console.WriteLine("\nAnother display using foreach\n");
34      foreach (double v in dsqrt){
35         Console.WriteLine("{0,7:e4}", v);
36      }
37
38      /*
39       *Two dimensional arrays
40       */
41
42      int[,] twod = new int[10, 3];
43
44      for(inti = 0; i<twod.GetLength(0); ++i) {
45         twod[i, 0] = i + 1;
46         twod[i, 1] = twod[i, 0] * twod[i, 0];
47         twod[i, 2] = twod[i, 1] * twod[i, 0];
48      }
49
50      for (inti = 0; i<twod.GetLength(0); ++i){
51         Console.WriteLine();
52         for (int j = 0; j <twod.GetLength(1); ++j){
53            Console.Write("{0,7:D}", twod[i, j]);
54         }
55      }
56
57      Console.WriteLine();
58      double[,] darray = {{0.1, 0.2}, {0.3,0.4}, {0.5, 0.6}};
59      for (inti = 0, j = 0; i<darray.GetLength(0); ++i){
60         Console.WriteLine("{0, -7:f2}{1, -7:f2}", darray[i, j], darray[i, j+1]);
61      }
62
63      //Jagged arrays
64
65      int[][] ijda = new int[][] { new int[] {1,2},
```

```
66                          new int[] {3,4,5},
67                         new int[]{6,7,8,9} };
68
69          Console.WriteLine("\nJagged array\n");
70          for(inti = 0; i<ijda.Length; ++i){
71             for (int j = 0; j <ijda[i].Length; ++j) {
72                Console.Write("{0, 4:d}", ijda[i][j]);
73             }
74             Console.WriteLine();
75          }
76
77          double[][] djda;
78          djda = new double[3][];
79          djda[0] = new double[5];
80          djda[1] = new double[2];
81          djda[2] = new double[7];
82
83          Random rd = new Random();
84          for(inti = 0; i<djda.Length; ++i){
85             for (int j = 0; j <djda[i].Length; ++j){
86                djda[i][j] = rd.NextDouble();
87             }
88          }
89
90          Console.WriteLine("\nAnother jagged array\n");
91          for (inti = 0; i<djda.Length; ++i){
92             for (int j = 0; j <djda[i].Length; ++j){
93                Console.Write("{0, 8:f4}", djda[i][j]);
94             }
95             Console.WriteLine();
96          }
97          Console.Read();
98       }
99    }
100}
```

Program 4: A program which ask the user for two values and displays all the prine numbers between those two values.

```
1 using System;
2
3 namespace PrimeNumbers{
```

```
4   class Program{
5      static public bool isPrime(intnum){
6         inti;
7         bool flag = true;
8         for (i = 2; i<= Math.Sqrt(num); ++i){
9            if (num % i == 0){
10               flag = false;
11               break;
12            }
13         }
14         return flag;
15      }
16
17      static void Main(string[] args){
18         int start, end, n, i, j;
19         System.Console.WriteLine("Enter the starting value of the range");
20         start = Int32.Parse(System.Console.ReadLine());
21         System.Console.WriteLine("Enter the ending value of the range");
22         end = Int32.Parse(System.Console.ReadLine());
23         for (i = start; i<= end; ++i){
24            if (isPrime(i)) {
25               System.Console.WriteLine(i);
26            }
27         }
28         System.Console.ReadLine();
29      }
30   }
31 }
```

1.7 Summary

This chapter gives the basic syntax of C#. It discusses about variables, keywords, data types, creation of arrays, operators, control structures, methods debugging and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

1.8 Review Questions

1) Write a note on .NET Framework.
2) Explain the data types in C#.
3) Explain the various operators in C#.
4) Discuss the various looping structures in C#.
5) Explain how arrays are created in C#.

6) What is a method? Explain its components.
7) How is debugging done in C#?

Reference

1) The Complete Reference: C#
2) Visual C# 2012: How to program.
3) https://docs.microsoft.com/en-us/dotnet/csharp/

CHAPTER 2: OOP with C#

Contents

2.0 Defining classes and class members

The **class** construct is the mechanism in C# for the OOP design paradigm of data encapsulation. A class is like a blueprint. It defines the data and behavior of a type. If the class is not declared as static, client code can create instances of it. These instances are **objects** which are assigned to a variable. The instance of a class remains in memory until all references to it go out of scope.The declaration and content of the class is shown below.

```
class C {
        ... fields, constants ...          //for object-oriented programming
        ... methods ...
        ... constructors, destructors ...

        ... properties ...                 //for component-based programming
        ... events ...

        ... indexers ...                   // for convenience
        ... overloaded operators ...

        ... nested types (classes, interfaces, structs, enums, delegates) ...
}
```

2.1 Constructors

Constructors are methods that are called when the object is firstcreated. To create an object, the constructor call is preceded by the keyword ‘new’. The process of doing this is calledinstantiation. An **object**is then referred to as an **instance** of its class.They are often used to initialize the data of an object.A constructor has the same as the name of its type (name ofclass). Its method signature includes only the method name andits parameter list; it does not include a return type.Objects are allocated on the heap (a memory region allocated for the progam). Objects must be created with new Eg. Stack stk = new Stack(50);

If you don't provide a constructor for your class, C# creates one by default that instantiates the object and sets member variables to the default values.If a constructor was declared, no default constructor is generated.

2.2 Destructors

Destructors/finalizers are used to destruct instances of classes.A class can only have one finalizer. Finalizers cannot be inherited or overloaded. Finalizers cannot be called. They are invoked automatically. A finalizer does not take modifiers or have parameters. The programmer has no control over when the finalizer is called because this is determined by the garbage collector.

A **field** is a variable of any type that is declared directly in a class or struct. Fields are members of their containing type.A class or struct may have instance fields or static fields or both. **Instance fields** are specific to an instance of a type. You can create two objects and modify the instance field in each object without affecting the value in the other object. By contrast, a **static field** belongs to the class itself, and is shared among all instances of that class. Changes made from object A will be visibly immediately to objects B and C if they access the field.

Fields are declared in the class block by specifying the access level of the field, followed by the type of the field, followed by the name of the field.Initialization of field is optional.Initialization value must becomputable at compile time. Field declared with '**const**' must beinitialized, value must be computable at compile time, and onceinitialized, its value cannot be changed throughout the program.

Fields specified with '**readonly**'keyword must be initialized in their declaration or in a constructor. Its value must not be changed later. The value need not be computable at compile time.

Variables that are a block of code or methods are called as **local variables**.

Scope is the region of your program within which the members and variables are accessible. The basic scope rules are as follows:

1. The scope of a parameter declaration is the body of the method in which the declaration appears.
2. The scope of a local-variable declaration is from the point at which the declaration appears to the end of the block containing the declaration.
3. The scope of a local-variable declaration that appears in the initialization section of a for statement's header is the body of the for statement and the other expressions in the header.
4. The scope of a method, property or field of a class is the entire body of the class. This enables non-static methods and properties of a class to use any of the class's fields, methods and properties, regardless of the order in which they're declared.

```
class Stack {
        int[] values;                   //Field
        int top = 0;                    //Field
```

```
        public Stack(int size) { ... }          //Constructor
        public void Push (int x) {...}          //method
        public intPop() {...}                   //method
        public ~Stack() { ... }                 //Destructor
}
```

To access any instance field or method of a class, we have to use the '**.**' (dot) operator. E.g. stk.Pop().

When a method is called, it is automatically passed a reference to the invoking object (that is, the object on which the method is called). This reference is called '**this**'. Therefore, this refers to the object on which the method is acting.

It is also possible to use 'this' inside a constructor. In this case, 'this' refers to the object that is being constructed. When the name of a parameter or a local variable is the same as the name of an instance variable, the local name hides the instance variable. You can gain access to the hidden instance variable by referring to it through 'this'. E.g.

```
class Rectangle {
        int x, y, width, height;
        public Rectangle (int x, int y, int w, int h) {this.x =x; this.y = y; width = w; height = h; }
        public Rectangle (int w, int h) : this(0, 0, w, h) {}
        public Rectangle () : this(0, 0, 0, 0) {}
                ...
}
```

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle(2, 5);
Rectangle r3 = new Rectangle(2, 2, 10, 5);
```

Constructors can be overloaded. A constructor may call another constructor with 'this'.Before a constructor is called, fields are possibly initialized.

2.3 Methods

Methods (called functions or procedures in other programming languages) allow you to modularize an app by separating its tasks into self-contained units. These methods are sometimes referred to as user-defined methods. The actual statements in the method bodies are written only once,can be reused from several locations in an app and are hidden from other methods. Methods give an advantage of the "divide-and-conquer" approach, which makes appdevelopment more

manageable by constructing apps from small, simple pieces. Another advantage is software reusability—existing methods can be used as building blocks to create new apps. A third advantage is avoid repeating code. Dividing an app into meaningful method makes the app easier to debug and maintain.

Arguments are passed to the methods in two ways. The first way is **call-by-value**. This method copies the value of an argument into the formal parameter of the method. Therefore, changes made to the parameter of the subroutine have no effect on the argument used in the call. By default, C# uses call-by-value.

The second way an argument can be passed is **call-by-reference**. In this method, a reference to an argument (not the value of the argument) is passed to the parameter. Inside the method, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will affect the argument used to call the method.

```
void Inc(int x) {x = x + 1;}
void F() {
        intval = 3;
        Inc(val);         // the value of val will still be 3
}
```

As just explained, value types, such as int or char, are passed by value to a method. This means that changes to the parameter that receives a value type will not affect the actual argument used in the call. If you want a method to be able to operate on the actual arguments that are passed to it then you can use of the **ref** and **out** keywords.

```
void Inc(ref int x) { x = x + 1; }
void F() {
        intval = 3;
        Inc(ref val);     // the value of val will be 4
}
```

```
void Read (out int x, out int y) {
        x = Console.Read(); y = Console.Read();
}
void F() {
        int first, next;
        Read(out first, out next);
}
```

An argument passed by ref must be assigned a value prior to the call. Thus, using ref, you cannot use a method to give an argument an initial value.
The method must assign the out parameter a value prior to the method's termination. Thus, after the call to the method, anout parameter will contain a value.

Methods of the same name can be declared in the same class, as long as they have different sets of parameters (determined by the number, types and order of the parameters).
This is called **method overloading**.When an overloaded method is called, the C# compiler selects the appropriate method by examining the number, types and order of the arguments in the call.Method overloading is commonly used to create several methods with the same name that perform the same or similar tasks, but on different types or different numbers of arguments. Methods of a class may be overloaded if they have different numbers of parameters, or if they have different parameter types, or if they have different parameter kinds (value, ref/out). Overloaded methods must not differ only in their function return types.
E.g.

```
void F (int x) {...}
void F (char x) {...}
void F (int x, long y) {...}
void F (long x, int y) {...}
void F (ref int x) {...}
intF() {...}
string F() {...}
```

Now if the variables are declared as below, then the appropriate method which are called is indicated.

```
inti; long n; short s;
F(i);          // F(int x)
F('a');        // F(char x)
F(i, n);       // F(int x, long y)
F(n, s);       // F(long x, int y);
F(i, s);       // ambiguous between F(int x, long y) and F(long x, int y); => compilation error
F(i, i);       // ambiguous between F(int x, long y) and F(long x, int y); => compilation error
F();           //ambiguous between intF() and string F();=> compilation error
```

A **static** variable represents classwide information—all objects of the class share the variable.

The scope of a static variable is the body of its class. A class's public static members can be accessed by qualifying the member name with the class name and the member access (.) operator. Static class members exist even when no objects of the class exist; they're available as soon as the class is loaded into memory at execution time.

A method declared static cannot access non-static class members directly, because a static method can be called even when no objects of the class exist. For the same reason, the 'this' reference cannot be used in a static method. Constants must not be declared as static.

```
class Rectangle {
        static Color defaultColor;      // once per class
        static readonlyint scale;       // once per class
        int x, y, width,height;         // once per object
        public static void ResetColor() {
                defaultColor = Color.white;
        }
        ...
}
```

To access static variables from other classes, use the class name instead of object name.
E.g.Rectangle.defaultColor ... Rectangle.ResetColor()...

2.4 Property

A **property** is declared like a field, but with a get/set block added. Properties look like fields from the outside, but internally they contain logic, like methods do.Property can have read-only and write-only fields. Property can validate a field when it is accessed.'get' and 'set' denote property accessors. The get accessor runs when the property is read. It must return a value of the property's type. The set accessor is run when the property is assigned. It has an implicit parameter named 'value' of the property's type that you typically assign to a private field.The most common implementation for a property is a getter and/or setter that simply reads and writes to a private field of the same type as the property. An automatic property declaration instructs the compiler to provide this implementation.

E.g.

```
public class Stock{
        decimal currentPrice;                   // The private "backing" field
        public decimal CurrentPrice{            // The public property
                get{
                        return currentPrice;
                }
                set{
                        currentPrice = value;
                }
        }
}
```

```
.....

Stock s = new Stock();
s.CurrentPrice = 100;  // Calls set
decimal v = s.CurrentPrice; // Calls get
```

E.g. Readonly property

```
class Account {
        long balance;
        public long Balance {
                get { return balance; }
        }
}
```

E.g. Automatic Property

```
public class Stock{
...
        public decimal CurrentPrice{ get; set; }
}
```

2.5 Indexers

Indexers provide a natural syntax for accessing elements in a class or struct that encapsulate a list or dictionary of values. Indexers are similar to properties, but are accessed via an index argument rather than a property name.A class may declare multiple indexers, each with parameters of different types. An indexer can also take more than one parameter.

E.g.

```
class Sentence{
        string[] words = "The quick brown fox".Split();
        public string this [intwordNum]{                        // indexer
                get{
                        return words [wordNum];
                }
                set{
                        words [wordNum] = value;
                }
        }
}

Sentence s = new Sentence();
```

```
Console.WriteLine (s[3]);       // fox
s[3] = "kangaroo";
Console.WriteLine (s[3]);       // kangaroo
```

E.g. Indexer having multiple parameters

```
public string this [int arg1, string arg2]{
        get { ... }  set { ... }
}
```

2.6 Operator Overloading

C# allows you to define the meaning of an operator relative to a class that you create. This process is called operatoroverloading. By overloading an operator, one expands its usage tothatclass. A principal advantage of operator overloading is that it allows you to seamlesslyintegrate a new class type into yourprogramming environment.

There are two forms of operator methods: one for unary operatorsand one for binary operators.

```
// General form for overloading a unary operator
public static ret-type operator op(param-type operand){
        // operations
}
// General form for overloading a binary operator
public static ret-type operator op(param-type1 operand1,     param-type1 operand2) {
        // operations
}
```

Operator methods must be both public and static.For unary operators, the operand must be of the same type as theclass for which theoperator is being defined. For binaryoperators, at least oneof the operands must be of the same typeas its class. Thus, you cannot overload any C# operators for objects that you have not created. For example, you can’tredefine + for int or string. Operator parameters must not use the ref or out modifier.

E.g.

```
class Fraction {
        int x, y;
        public Fraction (int x, inty){
                this.x = x;
                this.y = y;
        }

        public static Fraction operator +(Fraction a, Fraction b){
                return new Fraction(a.x * b.y + b.x * a.y, a.y* b.y);
```

```
        }
    }
```

Usage

```
    Fraction a = new Fraction(1, 2);
    Fraction b = new Fraction(3, 4);
    Fraction c = a + b;    // c.x == 10, c.y == 8
```

The following operators can be overloaded:
arithmetic: +, - (unary and binary), *, /, %, ++, --
relational: ==, !=, <, >, <=, >=
bit operators: &, |, ^
others: !, ~, >>, <<, true, false
Must always return a function result
If == (<, <=, true) is overloaded,!= (>=, >, false) must be overloaded as well.
To enable the use of the && and || short-circuit operators, you must follow four rules.
First, the class must overload & and |.
Second, the return type of the overloaded & and | methods must be the same as the class for which the operators are being overloaded.
Third, each parameter must be a reference to an object of the class for which the operator is being overloaded.
Fourth, the true and false operators must be overloaded for the class.
E.g.

```
    class TriState {
        intstate;                    // -1 == false, +1 == true, 0 == undecided
        public TriState(int s) { state = s; }
        public static bool operator true (TriState x) { returnx.state> 0; }
        public static bool operator false (TriState x) { returnx.state< 0; }
        public static TriState operator & (TriState x, TriState y) {
            if (x.state> 0 &&y.state> 0) return new TriState(1);
            else if (x.state< 0 || y.state< 0) return new TriState(-1);
            else return new TriState(0);
        }
        public static TriState operator | (TriState x, TriState y) {
            if (x.state> 0 || y.state> 0) return new TriState(1);
            else if (x.state< 0 &&y.state< 0) return new TriState(-1);
            else return new TriState(0);
        }
    }
```

2.7 Inheritance

Inheritance is one of the principles of OOP paradigm. It allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it.A class that is inherited is called a **base class**. The class that does the inheriting is called a **derived class**. It inherits all of the variables, methods, properties, andindexers defined by the base class and adds its own unique elements.

Using protected access offers an intermediate level of access between public and private. A base class's protected members can be accessed by members of that base class and by members of itsderived classes. Base-class members retain their original access modifier when they become members of the derived class. Methods of a derived class cannot directly access private members of the base class.

E.g.

```
class A {                       // base class
        int a;
        public A() {...}
        public void F() {...}
}

class B : A {                   // subclass (inherits from A, extends A)
        int b;
        public B() {...}
        public void G() {...}
}
```

B inherits a and F(), it adds b and G(). Constructors are not inherited. Inherited methods can be overridden. Single inheritance: a class can only inherit from one base class, but it can implement multiple interfaces. A class can only inherit from a class, not from a struct. Structs cannot inherit from another type, but they can implement multiple interfaces. A class without explicit base class inherits from Object.

The constructor for the base class constructs the base classportion of the object, and the constructor for the derived classconstructs the derived class part. The default constructorscreated automatically by C# and are called automatically from thetop level of the inhersitancehierarchy down to the constructor ofthe derived class object which is being created.When both the base class and the derived class define constructors, in this case, you must use another of C#'s keywords, base, which has two uses. The first use is to call a base class constructor. The second is to access a member of the base class that has been hidden by a member of a derived class.

E.g.

```
class A {// base class
        int a;
        public A(int a) {...}
        public void F() {...}
}

class B : A {                   // subclass (inherits from A, extends A)
        int b;
        public B(int x, int y) : A(x){...}
        public void G() {...}
}
```

A reference variable of a base class can be assigned a reference to an object of any class derived from that base class.

2.8 Method overriding

The process of redefining a virtual method (of a base class) inside a derived class is called **method overriding**.Only methods that are declared as virtual can be overridden insubclasses. Overriding methods must be declared as override.Method signatures must be identical same number and types ofparameters (including function type!) same visibility (public,protected, ...). Properties and indexers can also be overridden(virtual, override). Static and abstract methods cannot be overridden.

E.g.

```
class A {
        public  void F() {...}          // cannot be overridden
        public virtual  void G() {...}  // can beoverridden in subclasses
}

class B : A {
        public  void F() {...}          // warning: hides inherited F(). Use new
        public  void G() {...}          // warning: hidesinherited G(). Use new
        public overridevoid G() {       // ok: overridesinherited G
                ... base.G();           // callsinherited G()
        }
}
```

Method overriding forms the basis for one OOP principle of polymorphism. **Dynamic method dispatch** is the mechanism by which a call to an overridden method is resolved at runtime, rather than compile time. Dynamic method dispatch is how C# implements **runtime polymorphism**.

```
class A {
        public virtual void WhoAreYou() { Console.WriteLine("I am an A"); }
}

class B : A {
        public override void WhoAreYou() {
                Console.WriteLine("I am a B");
        }
}

A a = new B();
a.WhoAreYou();                  // "I am a B"

void Use (A x) {
        x.WhoAreYou();
}

Use(new A());                   // "I am an A"
Use(new B());                   // "I am a B"
```

2.9 Abstract class

An **abstract method** is created by specifying the abstract type modifier. An abstract method contains no body and is, therefore, not implemented by the base class. Thus, a derived class must override it; it cannot simply use the version defined in the base class. An abstract method is automatically virtual, and it is an error to use the virtual modifier. If a class has abstract methods (declared or inherited) it must be abstract itself. An **abstract class** has atleast one abstract method. One cannot create objects of an abstract class.

E.g.

```
abstract class Stream {
        public abstract void Write(char ch);
        public void WriteString(string s) {
                foreach (char ch in s)
                        Write(ch);
                }
        }
```

```
}
class File : Stream {
        public override void Write(char ch) {... write chto disk ...}
}
```

Classes declared with the key word '**sealed**' cannot be extended (same as final classes in Java) override methods can be declared as sealed individually. In that case, they cannot be overridden.
E.g.

```
sealed class Account : Asset {
        long balance;
        public void Deposit (long x) { balance += x; }
        public void Withdraw (long x) { balance -= x; }
        ...
}
```

3.10 Interface

An **interface** declaration begins with the keyword interface and can contain only abstract methods, abstract properties, abstract indexers, and abstract events. All interface members are implicitly declared both public and abstract. In addition, each interface can extend one or more other interfaces to create a more elaborate interface that other classes can implement. To use an interface, a class must specify that it implements the interface by listing the interface after the colon (:) in the class declaration. A concrete class implementing the interface must declare each member of the interface with the signature specified in the interface declaration. If a class which implements the interface does not define the body of all the methods of the interface, then that class is an abstract class and needs to be declared so.

Interfaces cannot have data members. They cannot defineconstructors, destructors, or operator methods. Also, no membercan be declared as static. You can declare a reference variable of an interface type. Such a variable can refer to any object that implements its interface. When you call a method on an object through an interface reference, it is the version of the method implemented by the object that is executed.One interface can inherit another.

E.g.

```
interface IList :ICollection, IEnumerable {
        int Add (object value);                 // methods
        bool Contains (object value);
        ...
        bool IsReadOnly{ get; }                 // property
        ...
        object  this [int index] { get; set; }  // indexer
}
```

```
class MyClass :MyBaseClass, IList, ISerializable {
        public int Add (object value) {...}
        public bool Contains (object value) {...}
        ...
        public bool IsReadOnly{ get {...} }
        ...
        public object this [int index] { get {...} set {...} }
}
IListobj = new MyClass();
obj.Add(…);
```

2.11 Struct and Class

A structure is similar to a class, but is a value type, rather than a reference type. Structures are declared using the keyword **struct** and are syntactically similar to classes, but with some differences.

Classes are reference types (objects are allocated on the heap). Struts are Value types (objects are allocated on the stack).
Classes support inheritance (all classes are derived from object). Structs do not support inheritance (but they are compatible with object).
Classes and Structs can implement interfaces.
Classes may declare a parameterless constructor. Structs must not declare a parameterless constructor.
Classes may have a destructor. Structs cannot have destructors.

2.12 Summary

This chapter gives the basic syntax of OOP in C#. It discusses about class, methods, constructors, destructor, method overloading and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

2.13 Review Questions

1) Explain OOP in C#.
2) Explain class and its member in C#.
3) Explain the methods in C#.
4) Explain constructor with example in C#.
5) Explain method overloading with example in C#.
6) Explain properties and indexer in C#?
7) Explain inheritance with example.
8) Explain method overriding with example.
9) Explain abstract class.
10) Explain Interface with example.
11) Explain structure with example.

Reference

1) The Complete Reference: C#
2) Visual C# 2012: How to program.
3) https://docs.microsoft.com/en-us/dotnet/csharp/

CHAPTER 3: Exception Handling

Contents

3.1 Exception Handling

An **exception** indicates that a problem occurred during a program's execution. The exception handling mechanism consists of try-catch blocks. A **try** statement specifies a code block subject to error-handling or cleanup code. The try block must be followed by a **catch** block, a finally block, or both. The catch block executes when an error occurs in the try block. The **finally** block executesafter execution leaves the try block (or if present, the catch block), to perform cleanup code, whether or not an error occurred. A catch block has access to an Exception object that contains information about the error. You use a catch block to either compensate for the error or **rethrow** the exception. Yourethrow an exception if you merely want to log the problem, or if you want to rethrow a new, higher-level exception type.A finally block adds determinism to your program, by always executing no matter what. It's useful for cleanup tasks such as closing network connections.

A try statement looks like this:

```
try{
        ... // exception may get thrown within execution of this block
}catch (ExceptionAex){
        ... // handle exception of type ExceptionA
}catch (ExceptionBex){
        ... // handle exception of type ExceptionB
}finally{
```

```
        ... // cleanup code
    }
```
E.g.
```
    FileStream s = null;
    try {
        s = new FileStream(curName, FileMode.Open);
        ...
    } catch (FileNotFoundException e) {
        Console.WriteLine("file {0} not found", e.FileName);
    } catch (IOException) {
        Console.WriteLine("some IO exception occurred");
    } catch {
        Console.WriteLine("some unknown error occurred");
    } finally {
        if (s != null) s.Close();
    }
```
The exception hierarchy is shown below:
```
Exception
    SystemException
        ArithmeticException
            DivideByZeroException
            OverflowException
            ...
        NullReferenceException
        IndexOutOfRangeException
        InvalidCastException
        ...
    IOException
        FileNotFoundException
        DirectoryNotFoundException
        ...
    WebException
    ...
    ApplicationException
        ... user-defined exceptions
        ...
```
Exception parameter name can be omitted in a catch clause. Exception type must be derived from System.Exception.If exception parameter is missing, System.Exception is assumed. The SytemException is thrown by the system; the IOException, WebException, etc are thrown by the library methods; ApplicationException are user defined exceptions thrown by the application.

3.2 Assembly

An **assembly** is the basic unit of deployment in .NET and is alsothe container for all types. An assembly contains compiled typeswith their IL code, runtime resources, and information to assistwith versioning, security, and referencing other assemblies. Anassembly also defines a boundary for type resolution and securitypermissioning. In general, an assembly comprises a single WindowsPortable Executable (PE) file—with an .exe extension in the caseof an application, or a .dll extension in the case of a reusablelibrary.

An assembly contains four kinds of things:

An **assembly manifest**: Provides information to the .NET runtime,such as the assembly's name, version, requested permissions, andother assemblies that it references.

An **application manifest**: Provides information to the operatingsystem, such as how the assembly should be deployed and whetheradministrative elevation is required.

Compiled types: The compiled IL code and metadata of the types defined within the assembly.

Resources: Other data embedded within the assembly, such asimages and localizable text.

Of these, only the assembly manifest is mandatory, although an assembly nearly always contains compiled types.

The Assembly ManifestThe assembly manifest serves two purposes:

It describes the assembly to the managed hosting environment.It acts as a directory to the modules, types, and resources inthe assembly.Assemblies are hence self-describing. A consumer can discover allof an assembly's data, types, and function, without needingadditional files. An assembly manifest is not something you addexplicitly to an assembly; it's automatically embedded into anassembly as part of compilation.Some of the functionally significant data stored in the manifestare: The simple name of the assembly, a version number (AssemblyVersion), a list of modules that comprise the assembly;a list of types defined in the assembly and the module containingeach type, etc.

The Application Manifest

An application manifest is an XML file that communicatesinformation about the assembly to the operating system. Anapplication manifest, if present, is read and processed beforethe .NET-managed hosting environment loads the assembly, and caninfluence how the operating system launches an application'sprocess.

Modules

The contents of an assembly are actually packaged within one ormore intermediate containers, called modules. A modulecorresponds to a file containing the contents of an assembly. The reason for this extra layer of containership is to allow anassembly to span multiple files which is useful feature whenbuilding an assembly containing code compiled in a mixture of programming languages.

Resource

An application typically contains not only executable code, butalso content such as text, images, or XML files. Such content canbe represented in an assembly through a resource.

Private assembly is used by only one application. It resides inthe application directory and it does not have a "strong name".Private assemblies cannot be signed.

Public assembly can be used by all applications. It resides inthe Global Assembly Cache (GAC) and must have a "strong name". It can be signed. GAC can hold assemblies with the same name butwith different version numbers.
A central repository is created on the computer during the installation of the .NET Framework for storing the .NET assemblies. This repository is called the **Global Assembly Cache** (GAC). GAC also contains a centralized copy of the .NET Framework itself. Versioning of assemblies in the GAC is centralized at the machine level and controlled by the administrator. GAC can improve startup time for very large assemblies, because the CLR verifies the signatures of the assemblies only once upon installation.

3.3 Garbage Collector

Every object which is created uses various system resources, such as memory. The CLR performs automatic memory management by using a **garbage collector** to reclaim the memory occupied by objects that are no longer in use. This is called garbage collection. The destructor is invoked by the garbage collector to perform termination housekeeping on an object before the garbage collector reclaims the object's memory.

Memory leaks, which are common in other languages (because memory is not automatically reclaimed in those languages), are less likely in C#. But the garbage collector is not guaranteed to perform its tasks at a specified time. Therefore, the garbage collector may call the destructor any time after the object becomes eligible for destruction, making it unclear when, or whether, the destructor will be called.

3.4 JIT Compiler

Before you can run Microsoft intermediate language (MSIL), itmust be compiled against the common language runtime to nativecode for the target machine architecture. The .NET Frameworkprovides two ways to perform this conversion: A .NET Framework**just-in-time (JIT) compiler** or The .NET Framework Ngen.exe(Native Image Generator).

Compilation by the JIT Compiler: JIT compilation converts MSIL to native code on demand atapplication run time, when the contents of an assembly are loadedand executed. Because the

common language runtimesupplies a JITcompiler for each supported CPU architecture, developers canbuild a set of MSIL assemblies that can be JIT-compiled and runon different computers with different machine architectures.However, if your managed code calls platform-specific native APIsor a platform-specific class library, it will run only on thatoperating system.
JIT compilation takes into account the possibility that some codemight never be called during execution. Instead of using time andmemory to convert all the MSIL in a PE file to native code, itconverts the MSIL as needed during execution and stores theresulting native code in memory so that it is accessible forsubsequent calls in the context of that process. The loadercreates and attaches a stub to each method in a type when thetype is loaded and initialized. When a method is called for thefirst time, the stub passes control to the JIT compiler, which converts the MSIL for that method into native code and modifies the stub to point directly to the generated native code. Therefore, subsequent calls to the JIT-compiled method go directly to the native code.The JIT compiler also enforces type-safety in the runtime environment of the .NET Framework. It checks for the values that are passed to parameters of any method.
The following are the various types of JIT compilation in .NET:
Pre - JIT.
Econo - JIT.
Normal - JIT.

Pre - JIT
In Pre-JIT compilation, complete source code is converted into native code in a single cycle (i.e. compiles the entire code into native code in one stretch). This is done at the time of application deployment. In .Net it is called "Ngen.exe"

Econo - JIT
In Econo-JIT compilation, the compiler compiles only those methods that are called at run time. After execution of this method the compiled methods are removed from memory.

Normal - JIT
In Normal-JIT compilation, the compiler compiles only those methods that are called at run time. After executing this method, compiled methods are stored in a memory cache. Now further calls to compiled methods will execute the methods from the memory cache.

3.5 Namespaces

A **namespace** defines a declarative region that provides a way to keep one set of names separate from another. In essence, names declared in one namespace will not conflict with the same names declared in another. The namespace used by the .NET Framework library (which is the C# library) is System.

Declaring a Namespace

A namespace is declared using the namespace keyword. The general form of namespace is:

```
namespace name {
    // members
}
```

Here, 'name' is the name of the namespace. A namespace declaration defines a scope. Anything declared immediately inside the namespace is in scope throughout the namespace. Within a namespace, you can declare classes, structures, delegates, enumerations, interfaces, or another namespace.If your program includes frequent references to the members of a namespace, having to specify the namespace each time you need to refer to a member quickly becomes tedious. The using directive removes this problem. **'using'** can also be employed to bring namespaces that you create into view.

E.g.

```
Color.cs
namespace Util {
    public enum Color {...}
}

Figure.cs
namespace Util.Figures {
    public class Rect {...}
    public class Circle {...}
}

Triangle.cs
namespace Util.Figures {
    public class Triangle {...}
}

Test.cs
using Util.Figures;
class Test {
    Rect r;         // without qualification (because of using Util.Figures)
    Triangle t;
    Util.Color c;   // with qualification
}
```

The using directive has a second form that creates another name,called an **alias**, for a type or a namespace. The syntax is:

```
using alias = name;
```

Here, alias becomes another name for the type (such as a class type) or namespace specified by name. Once the alias has been created, it can be used in place of the original name.
The namespace declaration can be split over several files or evenseparated within the same file. The contents of all the samenamespace will be in the scope of that namespace. In other words,once you use 'using' for the namespace, contents of all the declarations in all the files are available.
Namespace can be nested. Namespaces can be nested by more than two levels. When this is the case, a member in a nested namespace must be qualified with all of the enclosing namespace names.You can specify a nested namespace using a single namespace statement by separating each namespace with a period.If you don't declare a namespace for your program, then the **default global namespace** is used.
E.g.

```
File X.cs
namespace A {
        ... classes ...
        ... interfaces ...
        ... structs ...
        ... enumerations ...
        ... delegates ...

        namespace B {  // full name: A.B
                ...
        }
}

File Y.cs
namespace A {
        ...
        namespace B {...}
}
namespace C {...}
```

3.6 Summary

This chapter gives the basic syntax of exception handling in C#. It discusses about exception, Assembly, Components of Assembly, Private and Shared Assembly, Garbage Collector, JIT compiler, Namespaces and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

3.7 Review Questions

1. Write a short note on Assembly.
2. What is the significance of Assemblies in .NET?
3. List and Explain the Components of assemblies.
4. How Garbage collector works?
5. Explain JIT compiler.
6. What is namespace? Explain System namespace.

Reference

1) The Complete Reference: C#
2) Visual C# 2012: How to program.
3) https://docs.microsoft.com/en-us/dotnet/csharp/

CHAPTER 4: Collections, Comparisons and Conversions, Delegate and Events

Contents

4.0 Collections

In C#, collection represents group of objects. By the help of collections, we can perform various operations on objects such as

Collection types implement the following common functionality:

- Adding and inserting items to a collection
- Removing items from a collection
- Finding, sorting, searching items
- Replacing items
- Copy and clone collections and items
- Capacity and Count properties to find the capacity of the collection and number of items in the collection

All the data structure work can be performed by C# collections.

We can store objects in array or collection. Collection has advantage over array. Array has size limit but objects stored in collection can grow or shrink dynamically.

.NET supports two types of collections, generic collections and non-generic collections. Prior to .NET 2.0, it was just collections and when generics were added to .NET, generics collections were added as well.

The non-generic collections operate on data of type object. Thus, they can be used to store any type of data. They can be used to store any type of data, and different types of data can be mixed within the same collection.The non-generic collection classes and interfaces are in System.Collections.

The various interfaces are discussed briefly.
IEnumerable: Defines the GetEnumerator( ) method, which supplies the enumerator for acollection class. It provides the capabilityto loop through items in a collection.

ICollection: Defines the elements that all non-generic collections must have. Provides capability to obtain the number of items in a collection and copy items into a simple array type (inherits from IEnumerable).

IList: Defines a collection that can be accessed via an indexer (inherits from IEnumerable and ICollection).

IDictionary: Defines a collection that consists of key/value pairs (inherits from IEnumerable and ICollection).

4.1 Various Collection Classes and Their Usage

The following are the various commonly used classes of the System.Collection namespace. Click the following links to check their detail.

4.1.1 C# ArrayList

- ArrayList class is a collection that can be used for any types or objects.
- Arraylist is a class that is similar to an array, but it can be used to store values of various types.
- An Arraylist doesn't have a specific size.
- Any number of elements can be stored.

ArrayList Methods and Properties

- **Add()-** Add an object to a list.
- **Clear()**- Removes all the element from the list.
- **Contains()**–Determines if an element is in the list.
- **CopyTo()**- Copies a list to another.

- **Insert()**-Insert an element in to the list.
- **Remove()-** Removes the first occurrence of an element.
- **RemoveAt()**-Removes the element at specified field.
- **RemoveRange()**-Removes a range of element.
- **Sort()**-Sort the element.
- **Capacity**- Gets or sets the number of elements in the list.
- **Count** – get the number of elements currently in the list.

```
using System;
using System.Collections;
namespace ArrayListExample
{
   class Program
   {
      static void Main(string[] args)
      {
         // ArrayList Diclaration
         ArrayList name = new ArrayList();

         //Add method of array list

         name.Add("Amar");
         name.Add("Zaidi");
         name.Add("Saif");
         name.Add("Irfan");
         name.Add("Bhutik");
         name.Add("Zeeshan");
         name.Add("Pranali");
         name.Add("Sameer");

         // Capacity  and count property of arraylist
         Console.WriteLine("Capacity =   " + name.Capacity);
         Console.WriteLine("Element present = " + name.Count  + "\n");
         Console.WriteLine("element in the list\n\n");
         for (int i = 0; i < name.Count; i++)
         {
            Console.WriteLine(name[i]);
         }

         Console.WriteLine("element in the list after sorting\n\n");
```

```
            //sort method of arraylist
            name.Sort();
            for (int i = 0; i < name.Count; i++)
            {
               Console.WriteLine(name[i]);
            }
            Console.WriteLine("RemoveAt Method\n\n");
// removeat method
            name.RemoveAt(4);
            for (int i = 0; i < name.Count; i++)
            {
               Console.WriteLine(name[i]);
            }
            Console.WriteLine("Insert Method\n\n");
            name.Insert(4, "Mohaddesa");
            for (int i = 0; i < name.Count; i++)
            {
               Console.WriteLine(name[i]);
            }
            Console.WriteLine("\n\n contain method");
            Console.WriteLine("The element Mohaddesa contain in the arraylist is:" +
name.Contains("Mohaddesa"));
            Console.WriteLine("The element Razi contain in the arraylist is:" +
name.Contains("Razi"));
            Console.WriteLine("remove Method\n\n");
            name.Remove("Zaidi");
            for (int i = 0; i < name.Count; i++)
            {
               Console.WriteLine(name[i]);
            }
            Console.WriteLine("Reverse Method\n\n");
            name.Reverse();
            for (int i = 0; i < name.Count; i++)
            {
               Console.WriteLine(name[i]);
            }
            Console.WriteLine("Clear Method\n\n");
            Console.WriteLine("The number of element in the arraylist are:" + name.Count + "\n\n");
            name.Clear();
            Console.WriteLine("The number of element in the arraylist are:" + name.Count);
            Console.Read();

        }
    }
}
```

Example 2:

```
using System;
using System.Collections;

class Example{
   public static void Main(){
      // Create a new hash table.
HashtableopenWith = new Hashtable();

      // Add some elements to the hash table. There are no
      // duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");

      // The Add method throws an exception if the new key is already in the hash table.
try{
openWith.Add("txt", "winword.exe");
}catch{
Console.WriteLine("An element with Key = \"txt\"  already exists.");
      }

      // The Item property is the default property, so you
      // can omit its name when accessing elements.
Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

      // The default Item property can be used to change the value associated with a key.
openWith["rtf"] = "winword.exe";
Console.WriteLine("For key = \"rtf\", value = {0}.",openWith["rtf"]);

      // If a key does not exist, setting the default Item property
      // for that key adds a new key/value pair.
openWith["doc"] = "winword.exe";

      // ContainsKey can be used to test keys before inserting them.
      if (!openWith.ContainsKey("ht")){
openWith.Add("ht", "hypertrm.exe");
Console.WriteLine("Value added for key = \"ht\": {0}", openWith["ht"]);
      }
```

```
        // When you use foreach to enumerate hash table elements,
        // the elements are retrieved as KeyValuePair objects.
Console.WriteLine();
foreach( DictionaryEntry de in openWith ){
Console.WriteLine("Key = {0}, Value = {1}", de.Key, de.Value);
        }

        // To get the values alone, use the Values property.
ICollectionvalueColl = openWith.Values;

        // The elements of the ValueCollection are strongly typed
        // with the type that was specified for hash table values.
Console.WriteLine();
foreach( strings invalueColl ){
Console.WriteLine("Value = {0}", s);
        }

        // To get the keys alone, use the Keys property.
ICollectionkeyColl = openWith.Keys;

        // The elements of the KeyCollection are strongly typed
        // with the type that was specified for hash table keys.
Console.WriteLine();
foreach( strings inkeyColl ){
Console.WriteLine("Key = {0}", s);
        }

        // Use the Remove method to remove a key/value pair.
Console.WriteLine("\nRemove(\"doc\")");
openWith.Remove("doc");

        if (!openWith.ContainsKey("doc")){
Console.WriteLine("Key \"doc\" is not found.");
        }
    }
}

using System;
using System.Collections;
```

```
public class SamplesQueue{
   public static void Main(){
      // Creates and initializes a new Queue.
      Queue myQ = new Queue();
myQ.Enqueue("Hello");
myQ.Enqueue("World");
myQ.Enqueue("!");
      // Displays the properties and values of the Queue.
Console.WriteLine( "myQ" );
Console.WriteLine( "\tCount:    {0}", myQ.Count );
Console.Write( "\tValues:" );
PrintValues( myQ );
   }

   public static void PrintValues( IEnumerablemyCollection )  {
foreach( Objectobj in myCollection )
Console.Write( "    {0}", obj );
Console.WriteLine();
   }
 }

using System;
using System.Collections;
public class SamplesStack{
  public static void Main(){
     // Creates and initializes a new Stack.
     Stack myStack = new Stack();
myStack.Push("Hello");
myStack.Push("World");
myStack.Push("!");

     // Displays the properties and values of the Stack.
Console.WriteLine( "myStack" );
Console.WriteLine( "\tCount:    {0}", myStack.Count );
Console.Write( "\tValues:" );
PrintValues( myStack );
  }

  public static void PrintValues( IEnumerablemyCollection )  {
foreach( Objectobj in myCollection )
```

```
Console.Write( "    {0}", obj );
Console.WriteLine();
  }
}
```

4.1.2 C# HashTable

The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection.

A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Example

```
using System;
using System.Collections;
namespace ConsoleApplication6
{
   class Program
   {
      static void Main(string[] args)
      {
         Hashtable ht = new Hashtable();
         ht.Add("Zaidi", "C Sharp");
         ht.Add("Arif", "Java");
         ht.Add("Mohaddesa", "Oracle");
         ht.Add("Masoom", "Maths");

         foreach (DictionaryEntry d in ht)
   {
         Console.WriteLine(d.Key + " " + d.Value);
         Console.Read();

   }

      }
   }
}
```

4.2 Comparisons and Conversions

C# can convert between instances of compatible types. A conversion always creates a new value from an existing one. Conversions can be either implicit or explicit: implicit conversions happen automatically, and explicit conversions require a cast.

Implicit conversions are allowed when both of the following are true:

- The compiler can guarantee they will always succeed.
- No information is lost in conversion.

Conversely, explicit conversions are required when one of the following is true:

- The compiler cannot guarantee they will always succeed.
- Information may be lost during conversion.

When you cast from a floating-point number to an integral, any fractional portion is truncated; no rounding is performed. The static class System.Convert provides methods that round while converting between various numeric types.

4.2.1 Numeric Conversions

Integral to integral conversions

Integral conversions are implicit when the destination type can represent every possible value of the source type. Otherwise, an explicit conversion is required. For example:

```
int x = 5678;                   // int is a 32-bit integral
long y = x;                     // Implicit conversion to 64-bit integral
short z = (short)x;             // Explicit conversion to 16-bit integral
```

4.2.2 Floating-point to floating-point conversions

A float can be implicitly converted to a double, since a double can represent every possible value of a float. The reverse conversion must be explicit.

Floating-point to integral conversions

All integral types may be implicitly converted to all floating-point numbers:

```
inti = 4;
float f = i;
```

The reverse conversion must be explicit:

```
int i2 = (int)f;
```

Implicitly converting a large integral type to a floating-point type preserves magnitude but may occasionally lose precision. This is because floating-point types always have more magnitude than integral types, but may have less precision. Rewritingour example with a larger number demonstrates this:

```
int i1 = 200000001;
float f = i1;                   // Magnitude preserved, precision lost
int i2 = (int)f;                // 200000000
```

4.2.3 Decimal conversions

All integral types can be implicitly converted to the decimal type, since a decimal can represent every possible C# integral value. All other numeric conversions to and from a decimal type must be explicit.

Casts are optimized for efficiency; hence, they truncate data that won't fit. This can be a problem when converting from a real number to an integer, because often you want to round rather than truncate. Convert's numerical conversion methods address just this issue; they always round:

```
double d = 4.9;
int i = Convert.ToInt32 (d);                    // i == 5
int thirty = Convert.ToInt32("1E", 16);         // Parse in hexadecimal
uint five  = Convert.ToUInt32 ("101", 2);       // Parse in binary
```

Task	Functions	Examples

Parsing base 10 numbers	`Parse`
	`TryParse`
		`double d = double.Parse ("3.5");`
		`int i;`
		`bool ok = int.TryParse ("3", out i);`

Parsing from base 2, 8, or 16	`Convert.To`*`Integral`*	`int i = Convert.ToInt32 ("1E", 16);`
Formatting to hexadecimal	`ToString ("X")`	`string hex = 45.ToString ("X");`
Lossless numeric conversion	Implicit cast	`int i = 23;` `double d = d;`
Truncating numeric conversion	Explicit cast	`double d = 23.5;` `int i = (int) d;`
Rounding numeric conversion (real to integral)	`Convert.To`*`Integral`*	`double d = 23.5;` `int i = Convert.ToInt32 (d);`

The most common operations, String.Compare To and String.Equals or String.Equality use an ordinal comparison, a case-sensitive comparison. Ordinal comparisons will compare the strings character by character. Case-sensitive comparisons use capitalization in their comparisons.

The most important point about these default comparison methods is that because they use the current culture, the results depend on the locale and language settings of the machine where they run. These comparisons are unsuitable for comparisons where order should be consistent across machines or locations. The String.Equals method enables you to specify a StringComparison value of StringComparison.OrdinalIgnoreCase to specify a case-insensitive comparison. There is also a static Compare method that includes a boolean argument to specify case-insensitive comparisons.

Sometimes you will want to try a conversion at runtime, but not throw an exception if the conversion fails (which is the case when a cast is used). To do this, use the 'as' operator is used.expr as type

Here, expr is the expression being converted to type. If the conversion succeeds, then a reference to type is returned. Otherwise, a null reference is returned. The as operator canbe used to perform only reference, boxing, unboxing, or identity conversions.

Example:

```
using System;
namespace ConsoleApplication7
{
   class Program
   {
      static void Main(string[] args)
      {
         int i = 75;
         float f = 53.005f;
         double d = 2345.7652;
         bool b = true;

         Console.WriteLine(i.ToString());
         Console.WriteLine(f.ToString());
         Console.WriteLine(d.ToString());
         Console.WriteLine(b.ToString());
         Console.ReadKey();
      }
   }
}
```

4.3 Delegates

- A delegate dynamically wires up a method caller to its target method.
- There are two aspects to a delegate: type and instance.
- A delegate type defines a protocol to which the caller and target will conform, comprising a list of parameter types and a return type.
- A delegate instance is an object that refers to one (or more) target methods conforming to that protocol. A delegate instance literally acts as a delegate for the caller: the caller invokes the delegate, and then the delegate calls the target method.
- This indirection decouples the caller from the target method. A delegate type declaration is preceded by the keyword delegate, but otherwise it resembles an (abstract) method declaration.

Declaration of a delegate type

```
delegate void Notifier (string sender);          // ordinary method signature
                                                 // with the keyword delegate
```

Declaration of a delegate variable

```
Notifier greetings;
```

Assigning a method to a delegate variable

```
        void SayHello(string sender) {
                Console.WriteLine("Hello from " + sender);
        }
```

```
greetings = SayHello; // full form:     greetings = new Notifier(SayHello);
```

Calling a delegate variable

```
greetings("John");       // invokes SayHello("John") => "Hello from John"
```

Example

```
using System;
///A very basic example (SimpleDelegate1.cs):
namespace BasicDelegate
{
   // Declaration
   public delegate void SimpleDelegate();
   class TestDelegate
   {
      public static void MyFunc()
      {
         Console.WriteLine("I was called by delegate ...");
      }
      public static void Main()
      {
         // Instantiation
         SimpleDelegate obj = new SimpleDelegate(MyFunc);
         // Invocation
         obj();
         Console.Read();
      }
   }
}
```

4.3.1 Multicast Delegate

Multicast Delegate is an extension of normal delegates. It combines more than one method at a single moment of time.

Important fact about multicast delegate

In Multicasting, Delegates can be combined and when you call a delegate, a whole list of methods is called.

- All methods are called in FIFO (First in First Out) order.
- + or += Operator is used for adding methods to delegates.
- – or -= Operator is used for removing methods from the delegates list.

Example

```
using System;
namespace multicast_delegate_example
{
   delegate void MutiCastDelegate();
   class multiCastdelegateDemo
   {
      public static void Display()
      {
         Console.WriteLine("ZAIDI");
         Console.WriteLine();
      }
      public static void Print()
      {
         Console.WriteLine("RIZVI");
         Console.WriteLine();
      }

   }// class multiCastdelegateDemo close
   class Program
   {
      static void Main(string[] args)
      {
         MutiCastDelegate m1 = new MutiCastDelegate(multiCastdelegateDemo.Display);
         MutiCastDelegate m2 = new MutiCastDelegate(multiCastdelegateDemo.Print);
         MutiCastDelegate m3 = m1 + m2;
         MutiCastDelegate m4 = m2 + m1;
         MutiCastDelegate m5 = m3 - m2;
         m3();
         m4();
         m5();
         Console.Read();
      }
   }
}
```

4.4 Events

An **event** is, essentially, an automatic notification that some action has occurred. The control that generates an event is known as the **event sender**. When the user interacts with a GUI component, the interaction - known as an event - drives the program to perform a task. A method that performs a task in response to an event is called an **event handler**, and the overall process of responding to events is known as **event handling**. An object that has an interest in an event registers an event handler for that event. When the event occurs, all registered handlers are called. When the event occurs, the event sender calls its event handler to perform a task (i.e., to "handle the event"). We need a mechanism to indicate which method is the event handler for an event. Event handlers are represented by delegates.

Steps in Event Handling

Declare an event inside a class. (1)

To declare an event inside a class, first a delegate type for the event must be declared. (1a)

The event itself is declared. (1b)

Create method to raise the event. (2)

Create methods to handle the event. (3)

Hooking up to an event. (4)

Invoke/raise the event. (5)

Example

```
using System;
// Declare a delegate type for an event.
delegate void MyEventHandler();
// Declare a class that contains an event.
class MyEvent
{
   public event MyEventHandler SomeEvent;
   // This is called to raise the event.
   public void OnSomeEvent()
   {
      if (SomeEvent != null)
         SomeEvent();
   }
}
class EventDemo
{
   // An event handler.
   static void Handler()
   {
      Console.WriteLine("Event occurred");
   }
   static void Main()
   {
      MyEvent evt = new MyEvent();
      // Add Handler() to the event list.
      evt.SomeEvent += Handler;
      // Raise the event.
      evt.OnSomeEvent();
      Console.Read();

   }
}
```

4.4.1 Events in .NET

Events in the .NET Framework are based on the delegate model. The delegate model follows the observer design pattern, which enables a subscriber to register with, and receive notifications from, a provider. An event sender pushes a notification that an event has happened, and an event receiver receives that notification and defines a response to it.

Example

A System.Web.UI.WebControls.Button control raises an event when the user clicks it in the webpage. By handling the event, your application can perform the appropriate application logic for that button click.

To handle a button click event on a webpage:

1) Create a ASP.NET Web Forms page (webpage) that has a Button control with the OnClick value set to the name of method that you will define in the next step.

```
<asp:Button ID="Button1" runat="server" Text="Click Me" OnClick="Button1_Click" />
```

2) Define an event handler that matches the Click event delegate signature and that has the name you defined for the OnClick value.

```
protected void Button1_Click(object sender, EventArgs e){
        // perform action
}
```

3) In the event handler method that you defined in step 2, add code to perform any actions that are required when the event occurs.

4.5 Summary

This chapter gives the basic syntax of Collections, Comparisons and Conversions, Delegate and Events in C#. It discusses about ArrayList and various collection techniques also delegates and events and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

4.6 Exercise

1. What is collection? Explain collection in C#.
2. What is ArrayList? Explain with example.
3. Delegates in C# are used for Event Handling. Justify this statement with a relevant example program.
4. Write a program using any five Methods/Property of ArrayList Class.
5. Create a delegate with two int parameters and a return type. Create a class with two delegate methods multiply and divide. Write a program to implement the delegate.
6. What is delegate? Explain the steps to implement delegate in C#.NET.

Reference

1) The Complete Reference: C#
2) Visual C# 2012: How to program.
3) https://docs.microsoft.com/en-us/dotnet/csharp/
4) https://www.c-sharpcorner.com

CHAPTER 5 : Windows programming

Contents

5.0 Windows Programming

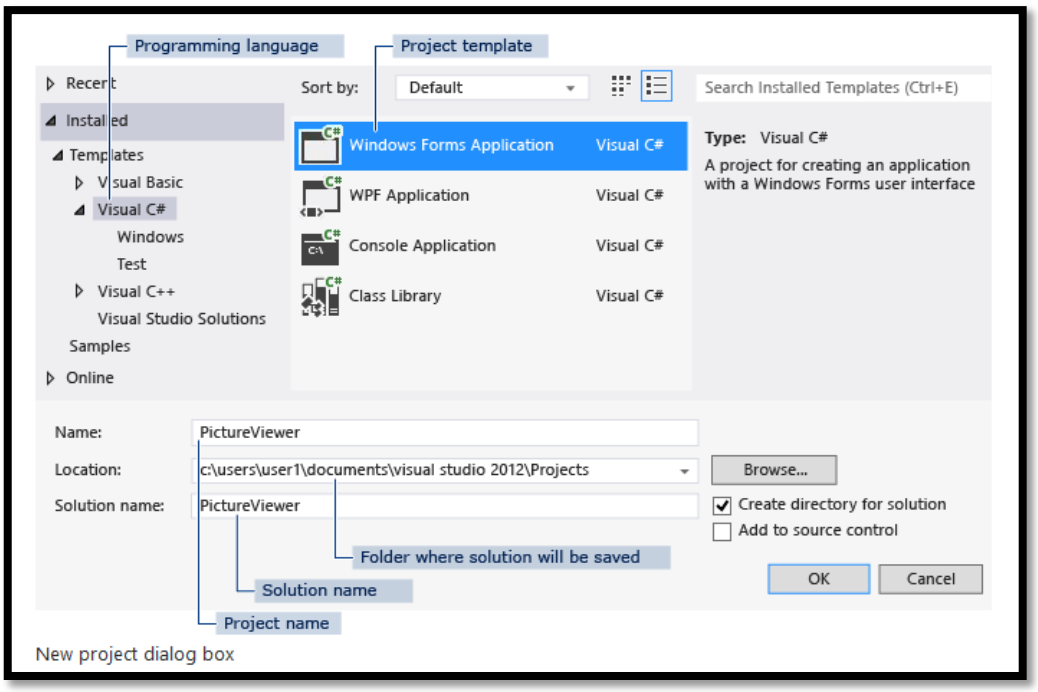
A graphical user interface (GUI) allows a user to interact visually with a program. GUIs are built from GUI controls. GUI controls are objects that can display information on the screen or enable users to interact with an app via the mouse, keyboard or some other form of input.

A Windows forms application is one that runs on the desktop computer. Windows Forms are used to create the GUIs for programs. A Form is a graphical element that appears on the desktop; it can be a dialog, a window or an MDI (multiple document interface) window. A Windows forms application will normally have a collection of controls such as labels, textboxes, list boxes, etc. A control has a graphical representation at runtime. A Form is a container for controls and components.

The steps to create a windows form application using Visual Studio are as follows:

1. On the menu bar, choose File, New, Project. The dialog box should look as shown in the figure.
2. Choose either Visual C# or Visual Basic in the Installed Templates list. We choose Visual C#.
3. In the templates list, choose the Windows Forms Application icon. Name the new form with your desired application name, and then choose the OK button. Visual Studio creates a solution for your program. A solution acts as a container for all of the projects and files needed by your program.

4. Now you are ready to add controls to the form and write event handlers.



5.1 Windows Controls

ASP.Net has the ability to add controls to a form such as textboxes and labels.

Windows Controls on a form is one of the interesting features of Visual Studio. This can be very useful actually, because we all know that creating Windows applications is sometime more flexible than Web applications.
In Windows applications you have more control and you have access to a lot of .NET Framework classes that you don't find in Web applications.

5.2 The Button Control

The .NET Framework provides a class derived from Control — System.Windows.Forms.ButtonBase— that implements the basic functionality needed in Button controls, so programmers can derive from this class and create their own custom Button controls.TheSystem.Windows.Forms namespace provides three controls that derive from ButtonBase: Button,CheckBox, and RadioButton.

A button is primarily used to perform three kinds of tasks:

- To close a dialog with a state (e.g., the OK and Cancel buttons)
- To perform an action on data entered in a dialog (e.g., clicking Search after entering somesearch criteria).
- To open another dialog or application (e.g., Help buttons)

The most frequently used event of a button is the Click event. This event happens whenever auser clicks the button, which means pressing the left mouse button and releasing it while the

pointer isover the button. Therefore, if one left-clicks the button and then draws the mouse away from the buttonbefore releasing it, the Click event will not be raised. In addition, the Click event is raised when thebutton has focus and the user presses the Enter key.

Common Button Class Properties:

PROPERTY	DESCRIPTION
FlatStyle	Changes the style of the button. If one set the style to Popup, the button appears flatuntil the user moves the mouse pointer over it. When that happens, the button pops up to a 3-D look.
Enabled	Although this is derived from Control, it's mentioned here because it's a very importantproperty for a button. Setting it to false means that the button becomes grayedout and nothing happens when one click it.
Image	Specifies an image (bitmap, icon, and so on) that will be displayed on thebutton.
ImageAlign	Specifies where the image on the button appears.

5.3 The Label And Linklabel Controls

The .NET Framework includes two label controls that are distinct:

- Label—The standard Windows label.
- LinkLabel—A label similar to the standard one (and derived from it) but that presents itself as an Internet link (a hyperlink).

The LinkLabel needs extra code to enable users clicking it to go to the target of the LinkLabel.

Common Label Control Properties:

PROPERTY	DESCRIPTION
BorderStyle	Specifies the style of the border around the label. The default is no border.
FlatStyle	Determines how the control is displayed. Setting this property to Popup makesthe control appear flat until the user moves the mouse pointer over the control,at which time the control appears raised.
Image	Specifies a single image (bitmap, icon, and so on) to be displayed in the label.
ImageAlign	Specifies where in the Label the image is shown.
LinkArea	Specifies the range in the text that should be displayed as alink.(LinkLabel only)
LinkColor	Indicates the color of the link. (LinkLabel only)
Links	It is possible for a LinkLabel to contain more than one link.This property enables one to find the link one want. The control keeps track ofthe links displayed in the text. Not available at design time.(LinkLabel only)
LinkVisited	Setting this to true means that the link is displayed in a differentcolor if it has been clicked.(LinkLabel only)
TextAlign	Specifies where in the control the text is shown.
VisitedLinkColor	Specifies the color of the LinkLabel after the user hasclicked it. (LinkLabel only)

5.4 The RadioButton and CheckBox Controls

The RadioButton and CheckBox controls share their base class with the Button control Radio buttons traditionally display themselves as a label with a tiny circle to the left of it, which can be either selected or not. One should use radio buttons when one wants to give users a choice between two or more mutually exclusive options — for example, undergraduate or graduate.

To group radio buttons together so they create one logical unit one must use a GroupBox control or some other container. One first places a GroupBox onto a form and then places the RadioButton controls one needs within the borders of the GroupBox, the RadioButton controls will automatically change their state to reflect that only one option within the group box can be selected. If one does not place the controls within a GroupBox, only one RadioButton on the *form* can be selected at any given time.

A CheckBox control traditionally displays itself as a label with a small box at its immediate left. Use a
check box when one wants to enable users to choose one or more options — for example, a questionnaire asking which programming languages the user has learnt (e.g., C,C++, Java and so on).

Common RadioButton Control Properties

PROPERTY	DESCRIPTION
Appearance	A radio button can be displayed either as a label with a circular check to the left, middle, or right of it, or as a standard button. When it is displayed as a button, the control appears pressed when selected, and not pressed otherwise.
AutoCheck	When true, a black point is displayed when the user clicks the radio button. When false, the radio button must be manually checked in code from the Click event handler.
CheckAlign	Used to change the alignment of the check box portion of the radio button. The default is ContentAlignment.MiddleLeft.
Checked	Indicates the status of the control. It is true if the control is displaying a black point, and false otherwise.

RadioButton Events

One will typically use only one event when working with RadioButton controls, but many others can be subscribed to.

Common RadioButton Control Events

EVENT	DESCRIPTION
CheckedChanged	Sent when the check of the RadioButton changes.
Click	Sent every time the RadioButton is clicked. This is not the same as the CheckedChange event, because clicking a RadioButton two or more times in succession changes the checked property only once — and only if it wasn't checked already. Moreover, if the AutoCheck property of the button being clicked is false, then the

button will not be checked at all, and only the Click event will be sent.

CheckBox Control Properties

PROPERTY	DESCRIPTION
CheckState	Unlike the radio button, a check box can have three states: Checked, Indeterminate, and Unchecked. When the state of the check box is Indeterminate, the control check next to the label is usually grayed out, indicating that either the current value of the check is not valid; for some reason cannot be determined (e.g., the check indicates the read-only state of files, and two are selected, of which one is read-only and the other is not); or has no meaning under the current circumstances.
ThreeState	When false, the user will not be able to change the CheckState state to Indeterminate. One can, however, still change the CheckState property to Indeterminate from code.

CheckBox Events

One will normally use only one or two events on this control. Although the CheckChanged event exists on both the RadioButton and the CheckBox controls, the effects of the events differ.

The GroupBox Control

The GroupBox control is often used to logically group a set of controls such as the RadioButton andCheckBox, and to provide a caption and a frame around this set.

CheckBox Control Events

EVENT	DESCRIPTION
CheckedChanged	Occurs whenever the Checked property of the check box changes. Note that in a CheckBox where the ThreeState property is true, it is possible to click the check box without changing the Checked property. This happens when the check box changes from Checked to Indeterminate status.
CheckStateChanged	Occurs whenever the CheckedState property changes. As Checked and Unchecked are both possible values of the CheckedState property, this event is sent whenever the Checked property changes. In addition, it is also sent when the state changes from Checked to Indeterminate.

To use the group box one drags it onto a form and then drags the controls it should contain onto it (but not the reverse — that is, one can't lay a group box over preexisting controls).The effect of this is that the parent of the controls becomes the group box, rather than the form, so it is possible to have more than one radio button selected at any given time. Within the group box, however, only one radio button can be selected. Moving the GroupBox control moves all of the controls placed on it. Another effect of placing controls on a group box is that it enables one to affect the contained controls by setting the corresponding property on the group box. For instance, if one wants to disable all the controls within a GroupBox control, one can simply set the Enabled property of the GroupBox to false.

5.5 The Textbox Control

Text boxes should be used whenever one wants users to enter text that one has no knowledge of atdesign time (e.g., the user's name). The primary function of a text box is for users to enter text, but any characters can be entered, and one can force users to enter numeric values only.

The .NET Framework comes with two basic controls to take text input from users: TextBox and RichTextBox. Both controls are derived from a base class called TextBoxBase, which itself is derivedfrom Control. TextBoxBase provides the base functionality for text manipulation in a text box, such as selecting text, cutting to and pasting from the clipboard, and a wide range of events.

Common TextBox Control Properties

PROPERTY	DESCRIPTION
CausesValidation	When a control with this property set to true is about to receive focus, two events are fired: Validating and Validated. One can handle these events in order to validate data in the control that is losing focus. This may cause the control never to receive focus.
CharacterCasing	A value indicating whether the TextBox changes the case of the text entered. Three values are possible: Lower: All text entered is converted to lowercase. Normal: No changes are made to the text. Upper: All text entered is converted to uppercase.
MaxLength	A value that specifies the maximum length, in characters, of any text entered into the TextBox. Set this value to zero if the maximum limit is limited only by available memory.
Multiline	Indicates whether this is a multiline control, meaning it is able to show multiple lines of text. When Multiline is set to true, one will usually want to set WordWrap to true as well.
PasswordChar	Specifies whether a password character should replace the actual characters entered into a single-lineTextBox. If the Multiline property is true, then this has no effect.
ReadOnly	A Boolean indicating whether the text is read-only.
ScrollBars	Specifies whether a multilineTextBox should display scroll bars.
SelectedText	The text that is selected in the TextBox.
SelectionLength	The number of characters selected in the text. If this value is set to be larger than the total number of characters in the text, then it is reset by the control to be the total number of characters minus the value of SelectionStart.
SelectionStart	The start of the selected text in a TextBox.
WordWrap	Specifies whether a multilineTextBox should automatically wrap words if a line exceeds the width of the control.

TextBox Events

TextBox Control Events

EVENT	DESCRIPTION
Enter ,Leave , Validating,Validated	These four events occur in the order in which they are listed here. Known as focus events, they are fired whenever a control's focus changes, with two exceptions. Validating and Validated are fired only if the control that receives focus has the CausesValidation property set to true. The receiving control fires the event because there are times when one do not want to validate the control, even if focus changes. An example of this is when a user clicks a Help button.
KeyDown, KeyPress KeyUp	These three are known as key events. They enable one to monitor and change what is entered into ones controls. KeyDown and KeyUp receive the key code corresponding to the key that was pressed. This enables one to determine whether special keys such as Shift or Ctrl and F1 were pressed. KeyPress, conversely, receives the character corresponding to a keyboard key. This means that the value for the letter 'a' is not the same as the letter 'A'. It is useful if one wants to exclude a range of characters — for example, only allowing numeric values to be entered.
TextChanged	Occurs whenever the text in the text box is changed, no matter what the change.

5.6 RichTextBox Control Properties:

PROPERTY	**DESCRIPTION**
CanRedo	True when the last undone operation can be reapplied using Redo.
CanUndo	True if it is possible to undo the last action on the RichTextBox. Note that CanUndo is defined in TextBoxBase, so it is available to TextBox controls as well.
RedoActionName	Holds the name of an action that would be performed by the Redo method.
DetectUrls	Set to true to make the control detect URLs and format them (underline, as in a browser).
Rtf	Corresponds to the Text property, except that this holds the text in RTF.
SelectedRtf	Use this to get or set the selected text in the control, in RTF. If one copies this text to another application — Word, for example — it will retain all formatting.
SelectedText	As with SelectedRtf, one can use this property to get or set the selected text. However, unlike the RTF version of the property, all formatting is lost.
SelectionAlignment	Represents the alignment of the selected text. It can be Center, Left, or Right.
SelectionBullet	Use this to determine whether the selection is formatted with a bullet in front of it, or use it to insert or remove bullets.

BulletIndent	Specifies the number of pixels a bullet should be indented.
SelectionColor	Changes the color of the text in the selection.
SelectionFont	Changes the font of the text in the selection.
SelectionLength	Set or retrieve the length of a selection.
SelectionType	Holds information about the selection. It will indicate whether one or more OLE objects are selected or if only text is selected.
ShowSelectionMargin	If true, a margin will be shown at the left of the RichTextBox. This makes it easier for the user to select text.
UndoActionName	Gets the name of the action that will be used if the user chooses to undo something.
SelectionProtected	One can specify that certain parts of the text should not be changed by setting this property to true.

RichTextBox Events

Most of the events used by the RichTextBox control are the same as those used by the TextBox control.

EVENT	**DESCRIPTION**
LinkClicked	Sent when a user clicks on a link within the text.
Protected	Sent when a user attempts to modify text that has been marked as protected.
SelectionChanged	Sent when the selection changes. If for some reason one does not want the user to change the selection, one can prevent the change here.

5.7 The Listbox And Checkedlistbox Controls

List boxes are used to show a list of strings from which one or more can be selected at a time. Just like check boxes and radio buttons, the list box provides a way to ask users to make one or more elections. One should use a list box when at design time one does not know the actual number of values from which the user can choose (e.g., a list of co-workers). Even if one knows all the possible values at design time, one should consider using a list box if there are a large number of values.

The ListBox class is derived from the ListControl class, which provides the basic functionality for list-type controls. Another kind of list box available is called CheckedListBox. Derived from the ListBox class, it provides a list just like the ListBox does, but in addition to the text strings it provides a check for each item in the list.

ListBox Properties

PROPERTY	**DESCRIPTION**
SelectedIndex	Indicates the zero-based index of the selected item in the list box. If the list boxcan contain multiple selections at the same time, then this property holds the index of the first item in the selected list.
ColumnWidth	Specifies the width of the columns in a list box with multiple columns.
Items	The Items collection contains all of the items in the list box. Oneuse the properties of this collection to add and remove items.

MultiColumn	A list box can have more than one column. Use this property to get or set whether values should be displayed in columns.
SelectedIndices	A collection that holds all of the zero-based indices of the selected items in the listbox.
SelectedItem	In a list box where only one item can be selected, this property contains the selected item, if any. In a list box where more than one selection can be made, it will contain the first of the selected items.
SelectedItems	A collection that contains all currently selected items.
SelectionMode	One can choose from four different modes of selection from the ListSelectionMode enumeration in a list box: None: No items can be selected. One: Only one item can be selected at any time. MultiSimple: Multiple items can be selected. With this style, when one click an item in the list it becomes selected and stays selected even if one click another item until one click it again. MultiExtended: Multiple items can be selected. One use the Ctrl, Shift, and arrows keys to make selections. Unlike MultiSimple, if one simply click an item and then another item afterwards, only the second item clicked is selected.
Sorted	When set to true, the ListBox alphabetically sorts the items it contains.
Text	One saw Text properties on a number of controls, but this one works differently from any one've seen so far. If one set the Text property of the ListBox control, it searches for an item that matches the text and selects it.

ListBox Methods

METHOD	DESCRIPTION
ClearSelected()	Clears all selections in the ListBox.
FindString()	Finds the first string in the ListBox beginning with a string one specify. For example, FindString("a")will find the first string in the ListBox beginning with 'a'.
FindStringExact()	Like FindString, but the entire string must bematched.
GetSelected()	Returns a value that indicates whether an item is selected.
SetSelected()	Sets or clears the selection of an item.
ToString()	Returns the currently selected item.
GetItemChecked()	(CheckedListBox only) Returns a value indicating whether an item is checked.
GetItemCheckState()	(CheckedListBox only) Returns a value indicating the check state of an item.
SetItemChecked()	(CheckedListBox only) Sets the item specified to a Checked state.
SetItemCheckState()	(CheckedListBox only) Sets the check state of an item.

ListBox Events

Normally, the events one will want to be aware of when working with a ListBox or CheckedListBox
are those related to the selections being made by the user.

EVENT	DESCRIPTION

ItemCheck	(CheckedListBox only) Occurs when the check state of one of the list items changes.
SelectedIndexChanged	Occurs when the index of the selected item changes.

5.8 The Listview Control

The list view is usually used to present data for which the user is allowed some control over the detail and style of its presentation. It is possible to display the data contained in the control as columns and rows much like in a grid, as a single column, or with varying icon representations. The most commonly used list view is like the one shown earlier, which is used to navigate the folders on a computer.

ListView Properties
PROPERTY DESCRIPTION

Property & Description
Alignment Gets or sets the alignment of items in the control.
AutoArrange Gets or sets whether icons are automatically kept arranged.
BackColor Gets or sets the background color.
CheckBoxes Gets or sets a value indicating whether a check box appears next to each item in the control.
CheckedIndices Gets the indexes of the currently checked items in the control.
CheckedItems Gets the currently checked items in the control.
Columns Gets the collection of all column headers that appear in the control.
GridLines Gets or sets a value indicating whether grid lines appear between the rows and columns containing the items and subitems in the control.
HeaderStyle Gets or sets the column header style.
HideSelection Gets or sets a value indicating whether the selected item in the control remains highlighted when the control loses focus.
HotTracking Gets or sets a value indicating whether the text of an item or subitem has the appearance of a hyperlink when the mouse pointer passes over it.
HoverSelection Gets or sets a value indicating whether an item is automatically selected when the mouse pointer remains over the item for a few seconds.
InsertionMark Gets an object used to indicate the expected drop location when an item is dragged within

a ListView control.
Items Gets a collection containing all items in the control.
LabelWrap Gets or sets a value indicating whether item labels wrap when items are displayed in the control as icons.
LargeImageList Gets or sets the ImageList to use when displaying items as large icons in the control.
MultiSelect Gets or sets a value indicating whether multiple items can be selected.
RightToLeftLayout Gets or sets a value indicating whether the control is laid out from right to left.
Scrollable Gets or sets a value indicating whether a scroll bar is added to the control when there is not enough room to display all items.
SelectedIndices Gets the indexes of the selected items in the control.
SelectedItems Gets the items that are selected in the control.
ShowGroups Gets or sets a value indicating whether items are displayed in groups.
ShowItemToolTips Gets or sets a value indicating whether ToolTips are shown for the ListViewItem objects contained in theListView.
SmallImageList Gets or sets the ImageList to use when displaying items as small icons in the control.
Sorting Gets or sets the sort order for items in the control.
StateImageList Gets or sets the ImageList associated with application-defined states in the control.
TopItem Gets or sets the first visible item in the control.
View Gets or sets how items are displayed in the control. This property has the following values: • LargeIcon − displays large items with a large 32 x 32 pixels icon. • SmallIcon − displays items with a small 16 x 16 pixels icon • List − displays small icons always in one column • Details − displays items in multiple columns with column headers and fields • Tile − displays items as full-size icons with item labels and sub-item information.
VirtualListSize Gets or sets the number of ListViewItem objects contained in the list when in virtual mode.
VirtualMode Gets or sets a value indicating whether you have provided your own data-management operations for the ListView control.

ListView Methods

- **BeginUpdate()** Tells the list view to stop drawing updates until EndUpdate() is called. This isuseful when one are inserting many items at once because it stops the view from flickering, and dramatically increases speed.
- **Clear()** Clears the list view completely. All items and columns are removed.
- **EndUpdate()** Call this method after calling BeginUpdate. When one call this method, the list view draws all of its items.
- **EnsureVisible()** Tells the list view to scroll itself to make the item with the index one specified visible.
- **GetItemAt()** Returns the ListViewItem at position x,y in the list view.

Events of the ListView Control

The following are some of the commonly used events of the ListView control −

Sr.No.	Event & Description
1	**ColumnClick** Occurs when a column header is clicked.
2	**ItemCheck** Occurs when an item in the control is checked or unchecked.
3	**SelectedIndexChanged** Occurs when the selected index is changed.
4	**TextChanged** Occurs when the Text property is changed.

5.9 TabControl Control

The TabControl manages tab pages where each page may host different child controls. In this article, I will demonstrate how to create and use a TabControl in Windows Forms.

The TabControl control provides an easy way to organize a dialog into logical parts that can beaccessed through tabs located at the top of the control. A TabControl contains TabPages that essentiallywork like a GroupBox control, in that they group controls together, although they are somewhatmore complex.
Using the TabControl is easy. One simply add the number of tabs one want to display to the control'scollection of TabPage objects and then drag the controls one want to display to the respectivepages.

TabControl Properties

The properties of the TabControl are largely used to control the appearance ofthe container of TabPage objects — in particular, the tabs displayed.

Understanding the TabControl and TabPage class

A TabControl is a collection of tab pages and a tab page is the actual control that hosts other child controls. TabPage class represents a tab page.

TabControl class represents a TabControl. This class provides members (properties, methods, and events) to work with the TabControls. Table 1 lists the TabControl properties.

Property	Description
Alignment	Area of the control where the tabs are aligned.
Appearance	Visual appearance of the control's tabs.
DrawMode	A way that the control's tab pages are drawn.
HotTrack	Value indicating whether the control's tabs change in appearance when the mouse passes over them.
ImageList	The images to display on the control's tabs.
ItemSize	Size of the control's tabs.
Multiline	A value indicating whether more than one row of tabs can be displayed.
Padding	Amount of space around each item on the control's tab pages.
RowCount	Returns the number of rows that are currently being displayed in the control's tab strip.
SelectedIndex	The index of the currently-selected tab page.
SelectedTab	Currently selected tab page.
ShowToolTips	The value indicating whether a tab's ToolTip is shown when the mouse passes over the tab.
SizeMode	The way that the control's tabs are sized.
TabCount	Number of tabs in the tab strip.
TabPages	Returns the collection of tab pages in this tab control.

Adding TabPage to a TabControl

Now I will add few tabs to the TabControl with the help of Properties window of TabControl. The Properties window has a property called TabPages, which is a collection of TabPage controls (see Figure 2). A TabPage represents a page of the TabControl that can host child controls.

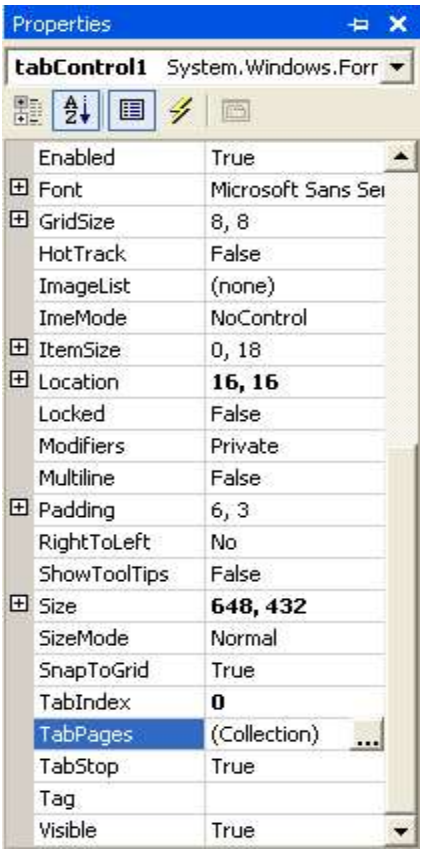


Figure 2. TabPages property of TabControl

Now if you click on TabPages property in Property window, it launches TabPage Collection Editor (see Figure 3) where you can add a new page or remove existing pages by using Add and Remove buttons. You can also set the properties of pages by using the right side properties grid. As you can see from Figure 3, I add two pages and set their properties.

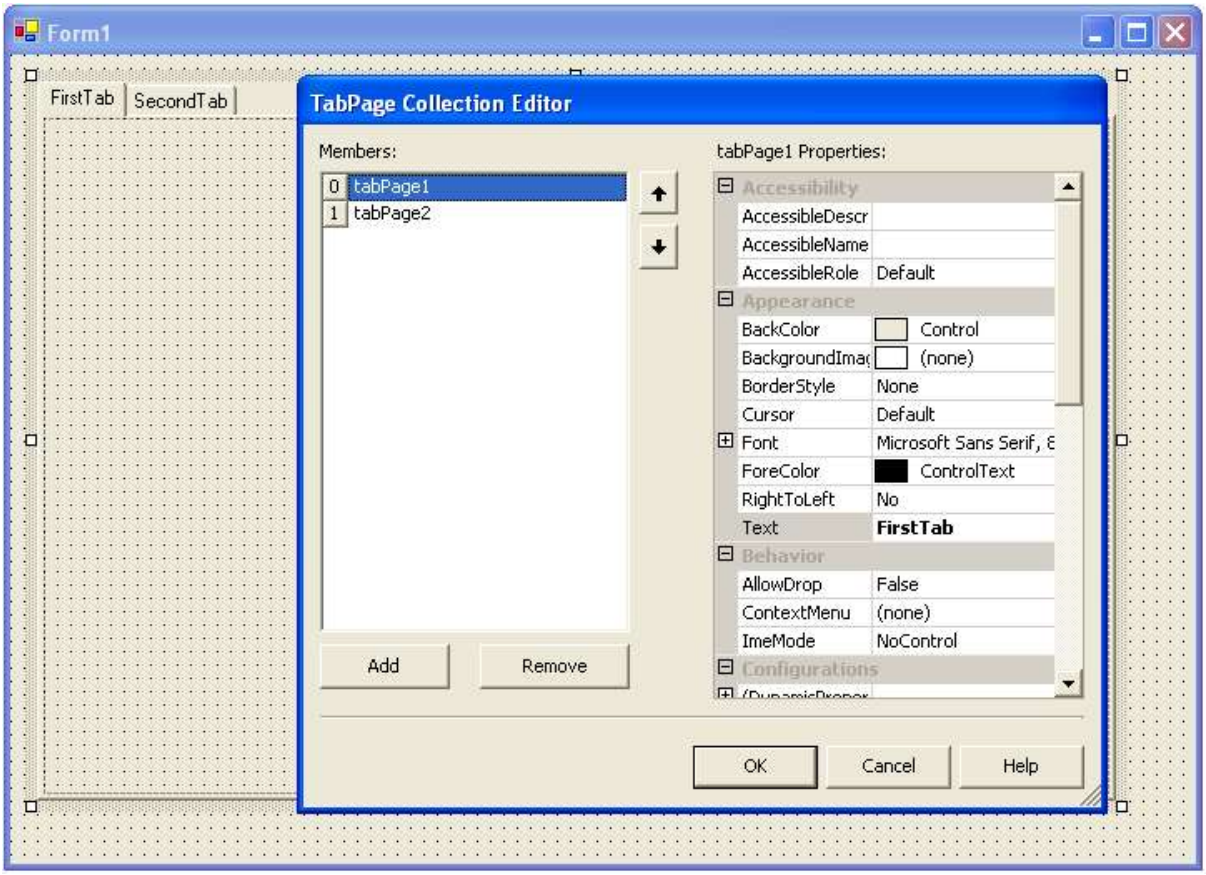


Figure 3. Adding Tab pages to a TabControl

After adding two pages to TabControl, the final Form looks like Figure 4.

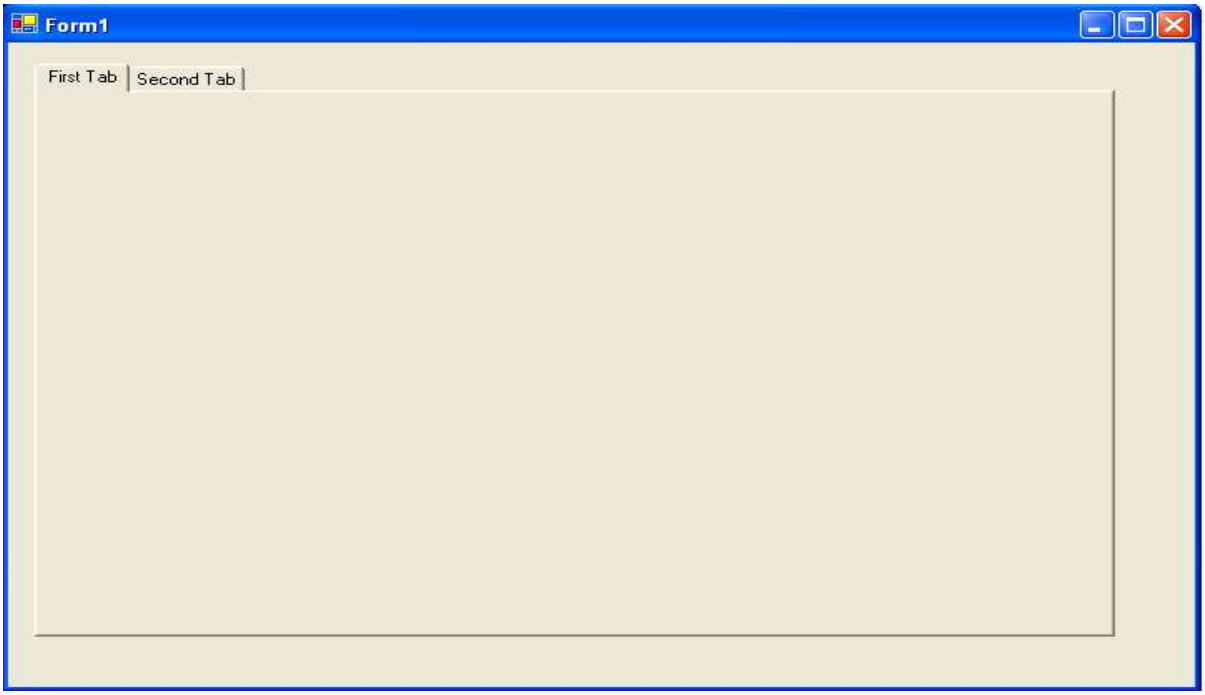


Figure 4. A Form with two Tab pages

Adding and Removing a TabPage to TabControl Programmatically

You can add and remove Tab pages to a TabControl using the TabControl.TabPages.Add and TabControl.TabPages.Remove methods. The following code snippet adds a new page to the TabControl programmatically:

```
1. TabPage newPage = new TabPage("New Page");
2. tabControl1.TabPages.Add(newPage);
```

After adding the page, the new TabControl would look like Figure 5.

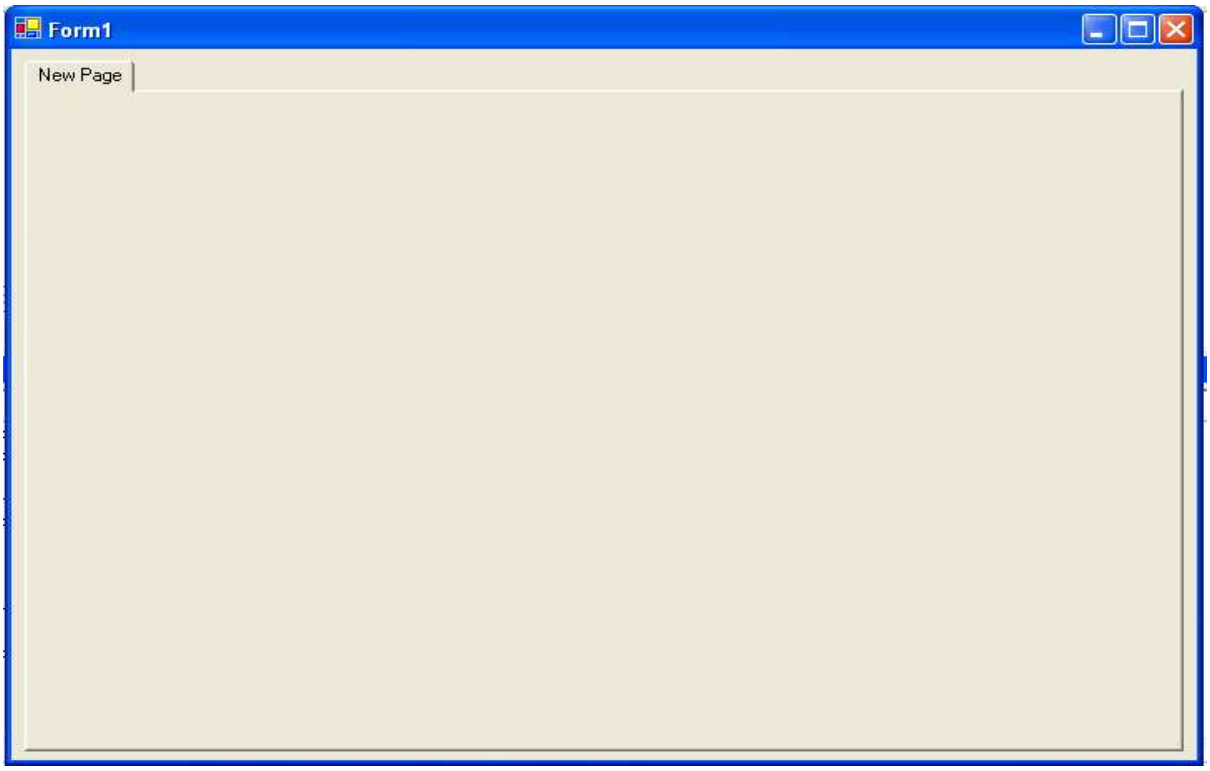


Figure 5.Adding a Tab page programmatically.

The Remove method of TabPageCollection class (through TabControl.TabPages) removes a page by name or index from the page collection. The following code snippet removes "New Page" from the collection:

```
1. TabPage newPage = new TabPage("New Page");
2. tabControl1.TabPages.Remove(newPage);
```

The RemoveAll method removes all the pages from the collection.

Adding Controls to a TabPage

Adding controls to a TabPage is similar to adding controls to a Form. Make a page active in the Form Designer and drag and drop controls from Toolbox to the page. I add a Label, a TextBox, and a Button control to Settings page of TabControl and change their properties. The final page looks like Figure 6.

Figure 6. Adding controls to a Tab page

Controls are added to a page by using TabPage.Controls.Add method. Now if you see the code generated by the designer, you will notice the following code:

```
1. this.SettingsPage.Controls.Add(this.BrowseBtn);
2. this.SettingsPage.Controls.Add(this.textBox1);
3. this.SettingsPage.Controls.Add(this.label1);
```

Using the same code, you can even add controls to a TabPage programmatically.

5.10 MENUS

How many Windows applications can one think of that do not contain a menu or toolbar of some Kind? None, right? Menus and toolbars are likely to be important parts of any application one will write for the Windows operating system. To assist one in creating them for one applications, Visual Studio 2010 provides two controls that enable one to create, with very little difficulty, menus and toolbars that look like the menus one find in Visual Studio.

The controls one will use can be grouped into a family of controls that has the suffix Strip. They are the ToolStrip, MenuStrip, and StatusStrip. One return to the StatusStrip later in the chapter. In their purest form, the ToolStrip and the MenuStrip are in fact the same control, because MenuStrip derives directly from the ToolStrip. This means that anything the ToolStrip can do, the MenuStrip can do.

The MenuStrip Control

In addition to the MenuStrip control, several additional controls are used to populate a menu. The three most common of these are the ToolStripMenuItem, ToolStripDropDown, and the ToolStripSeparator.

All of these controls represent a particular way to view an item in a menu or a toolbar.

The ToolStripMenuItem represents a single entry in a menu, the ToolStripDropDown represents an item that when clicked displays a list of other items, and the ToolStripSeparator represents a horizontal or vertical dividing line in a menu or toolbar.

There is another kind of menu that is discussed briefly after the discussion of the MenuStrip the ContextMenuStrip. A context menu appears when a user right-clicks on an item, and typically displays information relevant to that item.

A menu on a form is created with a Main Menu object, which is a collection of Menu Item objects. You can add menus to Windows Forms at design time by adding the Main Menu control and then adding menu items to it using the Menu Designer. Menus can also be added programmatically by adding one or more Main Menu controls to a form and adding Menu Item objects to the collection.

Now we perceive the following information regarding menus with instances.

1. Adding Menu,Menu Items to a Menu.
2. Adding Menu Enhancements to Windows Forms Menus.
3. Replacing, Cloning, Merging of Menus.
4. Context menus (Popupmenus).

Adding Menu, Menu Items to a Menu

First add a MainMenu control to the form. Then to add menu items to it add MenuItem objects to the collection. By default, a MainMenu object contains no menu items, so that the first menu item added becomes the menu heading. Menu items can also be dynamically added when they are created, such that properties are set at the time of their creation and addition.

The following program shows how to create a simple menu

The Class MenuTest1 creates a simple menu on a form. The form has a top-level File menu with menu items New,Open and Exit .The menu also includes a About menu.

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Threading.Tasks;
usingSystem.Windows.Forms;

namespace WindowsFormsApplication1
{
publicpartialclassForm1 : Form
   {
privateMainMenumainMenu;

publicvoid  MenuTest1()
      {
InitializeComponent();
mainMenu = newMainMenu();
MenuItem File = mainMenu.MenuItems.Add("&File");
File.MenuItems.Add(newMenuItem("&New"));
File.MenuItems.Add(newMenuItem("&Open"));
File.MenuItems.Add(newMenuItem("&Exit"));
this.Menu = mainMenu;
MenuItem About = mainMenu.MenuItems.Add("&About");
About.MenuItems.Add(newMenuItem("&About"));
this.Menu = mainMenu;
mainMenu.GetForm().BackColor = Color.Indigo;
      }

   }
}
```

5.11 TOOLBARS

While menus are great for providing access to a multitude of functionality in oner application, some items benefit from being placed in a toolbar as well as on the menu. A toolbar provides one-click access to such frequently used functionalities as Open, Save, and so on.

A button on a toolbar usually displays a picture and no text, although it is possible to have buttons with both. Examples of toolbars with no text are those found in Word, and examples of toolbars that include text can be found in Internet Explorer. In addition to buttons, onewill occasionally see combo boxes and text boxes in the toolbars too. If one let the mouse pointer hover above a button in a toolbar, it will often display a tooltip, which provides information about the purpose of the button, especially when only an icon is displayed.

The ToolStrip, like the MenuStrip, has been made with a professional look and feel in mind. When users see a toolbar, they expect to be able to move it around and position it wherever they want it. The ToolStrip enables users to do just that — that is, if one allow them to.

When one first add a ToolStrip to the design surface of oner form it looks very similar to the MenuStrip shown earlier, except for two things: To the far left are four vertical dots, just as one know them from the menus in Visual Studio. These dots indicate that the toolbar can be moved around and docked in the parent application window. The second difference is that by default a toolbar displays images, rather than text, so the default of the items in the bar is a button. The toolbar displays a drop-down menu that enables one to select the type of the item.

One thing that is exactly like the MenuStrip is that the Actions window includes a link called Insert Standard Items. When one click this, one don't get quite the same number of items as one did with the MenuStrip, but one get the buttons for New, Open, Save, Print, Cut, Copy, Paste, and Help.

ToolStrip Properties

The properties of the ToolStrip control and manage how and where the control is displayed. Remember that this control is actually the base for the MenuStrip control shown earlier, so many properties are shared between them. Again, the table that follows shows only a few properties of special interest — if one want a complete listing please refer to .NET Framework SDK documentation.

PROPERTY DESCRIPTION

GripStyle Controls whether the four vertical dots are displayed at the far left of the toolbar.

The effect of hiding the grip is that users can no longer move the toolbar.

LaonetStyle Controls how the items in the toolbar are displayed. The default is horizontally.=Items Contains a collection of all the items in the toolbar. ShowItemToolTipdetermines whether tooltips should be shown for the items in the toolbar.

Stretch By default, a toolbar is only slightly wider or taller than the items contained within it. If one set the Stretch property to true, the toolbar will fill the entire length of its container.

ToolStrip Items

One can use numerous controls in a ToolStrip. Earlier, it was mentioned that a toolbar should be able to contain buttons, combo boxes, and text boxes. As one would expect, there are controls for each of these items, but there are also quite a few others, described in the following table:

CONTROL DESCRIPTION

ToolStripButtonRepresents a button. One can use this for buttons with or without text. ToolStripLabelRepresents a label. It can also display images, which means that this control can be used to display a static image in front of another control that doesn't display information about itself, such as a text box or combo box.

ToolStripSplitButton Displays a button with a drop-down button to the right that, when clicked, displays a menu below it. The menu does not unfold if the button part of the control is clicked.

CONTROL DESCRIPTION

ToolStripDropDownButton Similar to the ToolStripSplitButton. The only difference is that the drop-down button has been removed and replaced with an image of a down arrow. The menu part of the control unfolds when any part of the control is clicked.

ToolStripComboBoxDisplays a combo box.

ToolStripProgressBar Embeds a progress bar in oner toolbar.

ToolStripTextBoxDisplays a text box.

ToolStripSeparatorCreates horizontal or vertical dividers for the items. One saw this control earlier.

5.12 SDI And MDI Applications

Traditionally, three kinds of applications can be programmed for Windows:

- **Dialog-based applications**: These present themselves to the user as a single dialog from whichall functionality can be reached.
- **Single-document interfaces (SDI)**: These present themselves to the user with a menu, one ormore toolbars, and one window in which the user can perform some task.
- **Multiple-document interfaces (MDI)**: These present themselves to the user in the same manneras an SDI, but are capable of holding multiple open windows at one time.

Dialog-based applications are usually small, single-purpose applications aimed at a specific task that needs a minimum of data to be entered by the user or that target a very specific type of data. An example of such an application is the Windows Calculator.

Single-document interfaces are each usually aimed at solving one specific task because they enable users to load a single document into the application to be worked on. This task usually involves a lot of user interaction, and users often want the capability to save or load the result of their work. Good examples of SDI applications are WordPad and Paint, both of which come with Windows. The simple text editor one've been creating in this chapter so far is another example of an SDI application.

However, only one document can be open at any one time, so if a user wants to open a second document, a fresh instance of the SDI application must be opened, and it will have no reference to the first instance. Any configuration one do to one instance is not carried over into the other. For example, in one instance of Paint one might set the drawing color to red, but when oneopen a second instance of Paint, the drawing color is the default, which is black.

Multiple-document interfaces are much the same as SDI applications, except that they can hold more than one document open in different windows at any given time. A telltale sign of an MDI applicationis the inclusion of the Window menu just before the Help menu on the menu bar. Visual Studio is anadvanced example of anMDI application. Every designer and editor in Visual

Studio opens in the same application, and the menus and toolbars adjust themselves to match the current selection.

5.13 Building MDI Applications

What is involved in creating an MDI? First, the task one want users to be able to accomplish should be one for which they would want to have multiple documents open at one time. A good example of this is a text editor or a text viewer. Second, one provide toolbars for the most commonly used tasks in the application, such as setting the font style, and loading and saving documents. Third, one provide a menu that includes a Window menu item that enables users to reposition the open windows relative to each other (tile and cascade) and that presents a list of all open windows. Another feature of MDI applications is that when a window is open and that window contains a menu, that menu should be integrated into the main menu of the application.
An MDI application consists of at least two distinct windows. The first window one creates is called an *MDI container*. A window that can be displayed within that container is called an *MDI child*. This chapter refers to the MDI container as the MDI container or main window interchangeably, and to the MDI child as the MDI child or child window.

You will create an MDI form in the WinApp project. You will also see how to create a menu bar for the parent form, that will allow you to navigate to all the child forms. To do so, follow these steps:

1. Navigate to Solution Explorer, select the WinApp project, right-click, and select "Add" -> "Windows form". Change the Name value from "Form1.cs" to "ParentForm.cs", and click "Add".
2. Select the newly added ParentForm in the Design View. Select the ParentForm form by clicking the form's title bar, navigate to the Properties window, and set the following properties:

    - Set the "IsMdiContainer" property to True (the default value is False). Notice that the background color of the form has changed to dark gray.
    - Set the Size property's Width to 546 and Height to 411.

3. Drag a MenuStrip control to the ParentForm. In the top-left corner, you should now see a drop-down showing the text "Type Here". Enter the text "Open Forms" in the drop-down. This will be your main, top-level menu.
4. Now under the Open Forms menu, add a submenu by entering the text "Win App".
5. Under the Win App submenu, enter "User Info".
6. Now click the top menu, "Open Forms", and on the right side of it, type "Help". Under the Help menu, enter "Exit".
7. Now, click the top menu, on the right side of Help, type "Windows".

8. Under the Windows menu, add the following options as separate submenus: Cascade, Tile Horizontal, Tile Vertical, and Arrange Icons. These will help in arranging the child forms.
9. Now it's time to attach code to the submenus you have added under the main menu Open Forms. First, you'll add code for the submenu Win App, that basically will open the WinApp form. In the Design View, double-click the "Win App" submenu, that will take you to the Code View. Under the click event, add the following code:

```
WinAppobjWA = new WinApp();
objWA.Show();
```

10. Now to associate functionality with the User Info submenu: double-click this submenu, and under the click event add the following code:

```
UserInfoobjUI = new UserInfo();
objUI.Show();
```

11. To associate functionality with the Exit submenu located under the Help main menu, double-click "Exit", and under the click event add the following code:

```
Application.Exit();
```

12. Now you have the form-opening code functionality in place, and you are nearly set to run the application. But first, you need to set the ParentForm as the start-up object. To do so, open Program.cs, and modify the "Application.Run(new UserInfo());" statement to the following:

```
Application.Run(new ParentForm());
```

13. Now build the solution, and run the application by pressing F5; the MDI application will open and should look as in Figure 1-1.

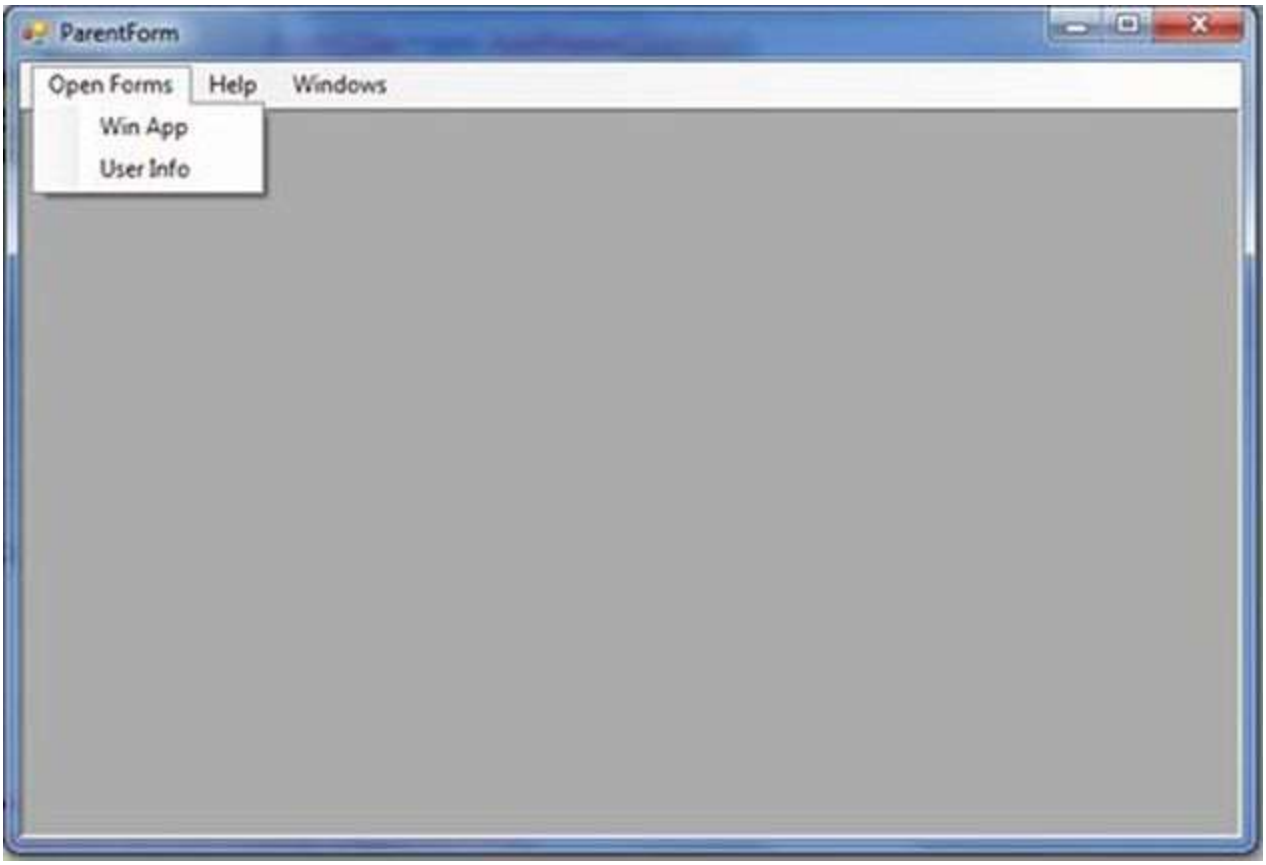


Figure 1-1. Running an MDI form application

14. Now if you click "Win App" and then "User Info" then both the forms will open one by one. These forms can be opened and dragged outside of the MDI form. This is not an expected behavior from a MDI application, as shown in Figure 1-2.

This issue will be addressed later in this chapter.

Figure 1-2. Running an MDI form application

How It Works

Each Windows Forms form is a class and exposes a Show() function by an instance created for it. You use the following code, that is creating an object and then invoking the Show() method. This opens the other form from the MDI parent form.

This creates an instance of the WinApp form and opens it for you:

```
WinAppobjWA = new WinApp();
objWA.Show();
```

The following code creates an instance of the UserInfo form and opens it for you:

```
UserInfoobjUI = new UserInfo();
objUI.Show();
```

You close the application with the following code:

```
Application.Exit();
```

5.14 Summary

This chapter gives the basic of Windows form control in asp,net. It discusses about controls properties, methods and events. After learning the above topics, you can desing windows

application and many useful programs and built a strong foundation for larger programming projects.

5.15 Exercise

1. Explain the text box control. List and explain any four text box attributes.
2. What is the difference between check box and radio button control? What are the common attributes associated with these controls?
3. Explain any five properties of List box and Drop-down list controls.
4. Explain Listbox with properties and methods.
5. What is the difference between List Box and Drop-Down Lists? List and explain any three common properties of these controls.
6. Explain MDI Form in details
7. Explain Menu control with its properties.
8. Explain Toolbar with its properties.

Reference

1) The Complete Reference: C#
2) Visual C# 2012: How to program.
3) https://docs.microsoft.com/en-us/dotnet/csharp/
4) https://www.c-sharpcorner.com

Introduction to ASP.Net 4

The .NET Framework is Microsoft's comprehensive and consistent programming model for building applications that have visually stunning user experiences, seamless and secure communication, and the ability to model a range of business processes.

The .NET Framework 4 works side by side with older Framework versions. Applications that are based on earlier versions of the Framework will continue to run on the version targeted by default.

The Microsoft .NET Framework 4 provides the following new features and improvements:

- Improvements in Common Language Runtime (CLR) and Base Class Library (BCL)
    - Performance improvement including better multicore support, background garbage collection, and profiler attach on server.
    - New memory mapped file and numeric types.
    - Easier debugging including dump debugging, Watson minidumps, mixed mode debugging for 64 bit and code contracts.
- Innovations in the Visual Basic and C# languages, for example statement lambdas, implicit line continuations, dynamic dispatch, and named/optional parameters.
- Improvements in Data Access and Modeling
    - The Entity Framework enables developers to program against relational databases using .NET objects and Language Integrated Query (LINQ). It has many new features, including persistence ignorance and POCO support, foreign key associations, lazy loading, test-driven development support, functions in the model, and new LINQ operators. Additional features include better n-tier support with self-tracking entities, customizable code generation using T4 templates, model first development, an improved designer experience, better performance, and pluralization of entity sets. For more information go here.
    - WCF Data Services is a component of the .NET Framework that enables you to create REST-based services and applications that use the Open Data Protocol (OData) to expose and consume data over the Web. WCF Data Services has many new features, including enhanced BLOB support, data binding, row count, feed customization, projections, and request pipeline improvements. Built-in integration with Microsoft Office 2010 now makes it possible to expose Microsoft Office SharePoint Server data as an

OData feed and access that data feed by using the WCF Data Services client library

- Enhancements to ASP.NET
    - More control over HTML, element IDs and custom CSS that make it much easier to create standards-compliant and SEO-friendly web forms.
    - New dynamic data features including new query filters, entity templates, richer support for Entity Framework 4, and validation and templating features that can be easily applied to existing web forms.
    - Web forms support for new AJAX library improvements including built-in support for content delivery networks (CDNs).
- Improvements in Windows Presentation Foundation (WPF)
    - Added support for Windows 7 multi-touch, ribbon controls, and taskbar extensibility features.
    - Added support for Surface 2.0 SDK.
    - New line-of-business controls including charting control, smart edit, data grid, and others that improve the experience for developers who build data centric applications.
    - Improvements in performance and scalability.
    - Visual improvements in text clarity, layout pixel snapping, localization, and interoperability.
- Improvements to Windows Workflow (WF) that enable developers to better host and interact with workflows. These include an improved activity programming model, an improved designer experience, a new flowchart modeling style, an expanded activity palette, workflow-rules integration, and new message correlation features. The .NET Framework 4 also offers significant performance gains for WF-based workflows.
- Improvements to Windows Communication Foundation (WCF) such as support for WCF Workflow Services enabling workflow programs with messaging activities, correlation support. Additionally, .NET Framework 4 provides new WCF features such as service discovery, routing service, REST support, diagnostics, and performance.
- Innovative new parallel-programming features such as parallel loop support, Task Parallel Library (TPL), Parallel LINQ (PLINQ), and coordination data structures which let developers harness the power of multi-core processors.

ASP.NET Lifecycle

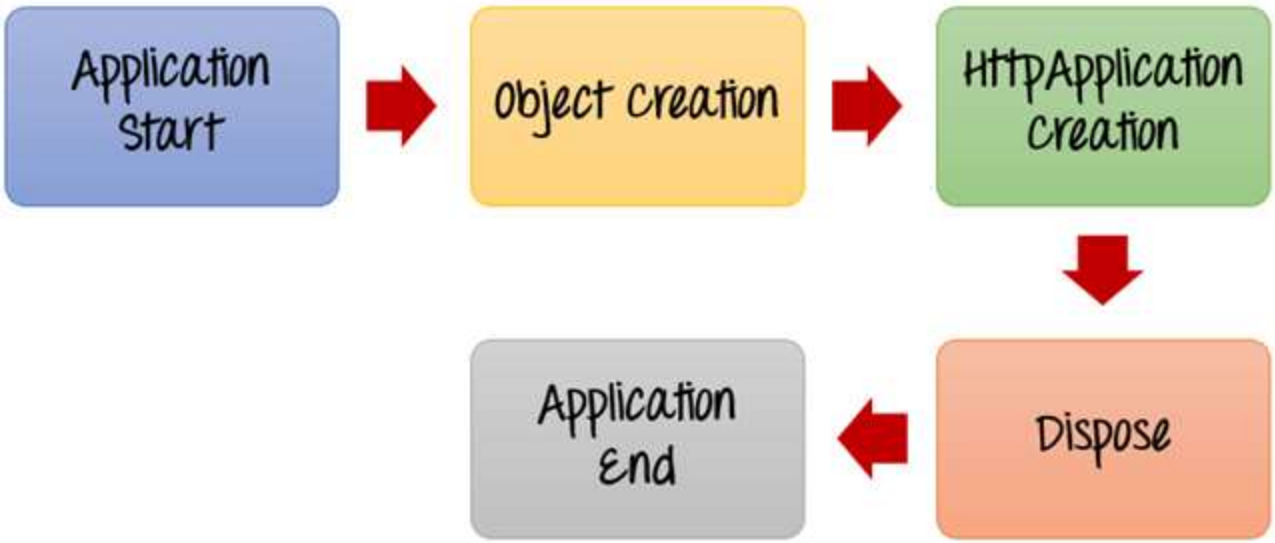
ASP.NET life cycle specifies, how:

- ASP.NET processes pages to produce dynamic output.
- The application and its pages are instantiated and processed.
- ASP.NET compiles the pages dynamically.

The ASP.NET life cycle could be divided into two groups:

- Application Life Cycle.
- Page Life Cycle.

Let's look at the various stages of a typical page lifecycle of an ASP.Net Web Application.



1. Application Start: The life cycle of an ASP.NET application starts when a request is made by a user. This request is to the Web server for the ASP.Net Application. This happens when the first user normally goes to the home page for the application for the first time. During this time, there is a method called Application_start which is executed by the web server. Usually, in this method, all global variables are set to their default values.

2. Object Creation: The next stage is the creation of the HttpContext, HttpRequest & HttpResponse by the web server. The HttpContext is just the container for the HttpRequest and HttpResponse objects. The HttpRequest object contains information about the current request, including cookies and browser information. The HttpResponse object contains the response that is sent to the client.

3. HttpApplication Creation: This object is created by the web server. It is this object that is used to process each subsequent request sent to the application. For example, let's assume we have 2 web applications. One is a shopping cart application, and the other is a news website. For each application, we would have 2 HttpApplication objects created. Any further requests to each website would be processed by each HttpApplication respectively.

4. Dispose: This event is called before the application instance is destroyed. During this time, one can use this method to manually release any unmanaged resources.

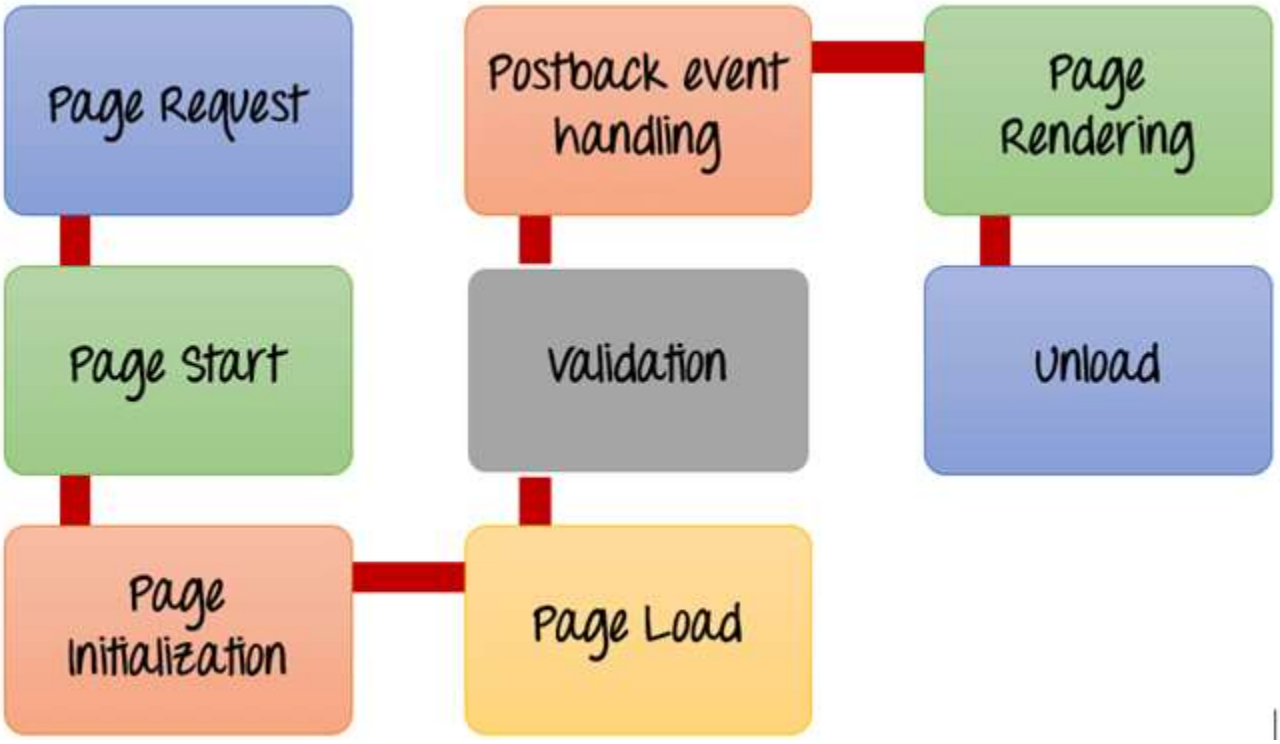
5. Application End: This is the final part of the application. In this part, the application is finally unloaded from memory.

ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

Let's look at the various stages of the lifecycle of an ASP.Net web page.



1. Page Request: This is when the page is first requested from the server. When the page is requested, the server checks if it is requested for the first time. If so, then it needs to compile the page, parse the response and send it across to the user. If it is not the first time the page is requested, the cache is checked to see if the page output exists. If so, that response is sent to the user.

2. Page Start: During this time, 2 objects, known as the Request and Response object are created. The Request object is used to hold all the information which was sent when the page was requested. The Response object is used to hold the information which is sent back to the user.

3. Page Initialization: During this time, all the controls on a web page is initialized. So if you have any label, textbox or any other controls on the web form, they are all initialized.

4. Page Load: This is when the page is actually loaded with all the default values. So if a textbox is supposed to have a default value, that value is loaded during the page load time.

5. Validation: Sometimes there can be some validation set on the form. For example, there can be a validation which says that a list box should have a certain set of values. If the condition is false, then there should be an error in loading the page.

6. Postback Event Handling: This event is triggered if the same page is being loaded again. This happens in response to an earlier event. Sometimes there can be a situation that a user clicks on a submit button on the page. In this case, the same page is displayed again. In such a case, the Postback event handler is called.

7. Page Rendering: This happens just before all the response information is sent to the user. All the information on the form is saved, and the result is sent to the user as a complete web page.

8. Unload: Once the page output is sent to the user, there is no need to keep the ASP.net web form objects in memory. So the unloading process involves removing all unwanted objects from memory.

Let’s look at the ASP.NET life cycle events in detail:

Page Event	Typical Use
PreInit	This event is raised after the start stage is complete and before the initialization
Init	This event occurs after all controls have been initialized. We can use this event to read or initialize control properties.
InitComplete	This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sur after the next postback.
PreLoad	This event is occurs before the post back data is loaded in the controls.
Load	This event is raised for the page first time and then recursively for all child cont

Control events	This event is used to handle specific control events such as Button control' Click
LoadComplete	This event occurs at the end of the event-handling stage. We can use this event for tasks that require all other controls on the page be lo
PreRender	This event occurs after the page object has created all controls that are require render the page.
PreRenderComplete	This event occurs after each data bound control whose DataSourceID property i DataBind method.
SaveStateComplete	It is raised after view state and control state have been saved for the page and
Render	This is not an event; instead, at this stage of processing, the Page object calls t each control.
Unload	This event raised for each control and then for the page.

ASP.NET Server Controls

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

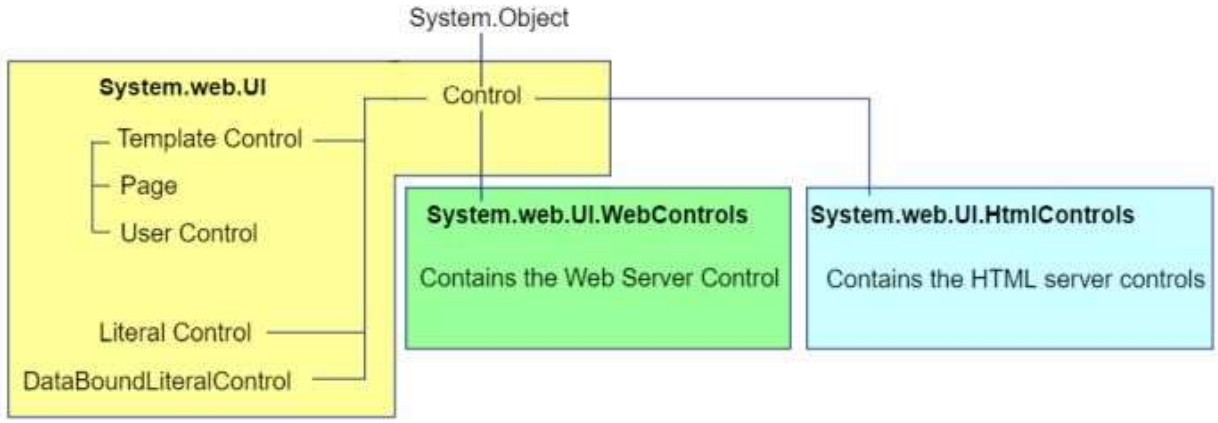
These controls provide the following features:

- Automatic state management.
- Simple access to object values without having to use the Request object.
- Ability to react to events in server-side code to create applications that are better structured.
- Common approach to building user interfaces for Web pages.
- Output is automatically customized based on the capabilities of the browser.

In addition to the built-in controls, the ASP.NET page framework also provides the ability to create user controls and custom controls. User controls and custom controls can enhance and extend existing controls to build a much richer user interface.

Hierarchy of Server Control :

Control class in System.Web.UI namespace is the base class of all the web server controls. In fact, page class which is the base class of web form we create, is also derived from Control class.



Control Class Properties:

ClientID :

Return unique identifier for the control created by ASP.NET when the page is instantiated.

Controls :

Return child controls collection.

EnableViewState :

Returns the boolean value indicating whether viewstate should be maintained across postback.

Visible :

Returns boolean value indicating whether control should be displayed or not.

Parent :

Returns reference to control which is parent to the control.

Page :

Returns reference to page which contains control.

ID:

Returns or sets identifier for the control.

Control Class Methods:

DataBind( ):

Bind Control to its data source.

FindControl( ):

Searches and returns child control within parent control. Returned control needs to be cast to proper type.

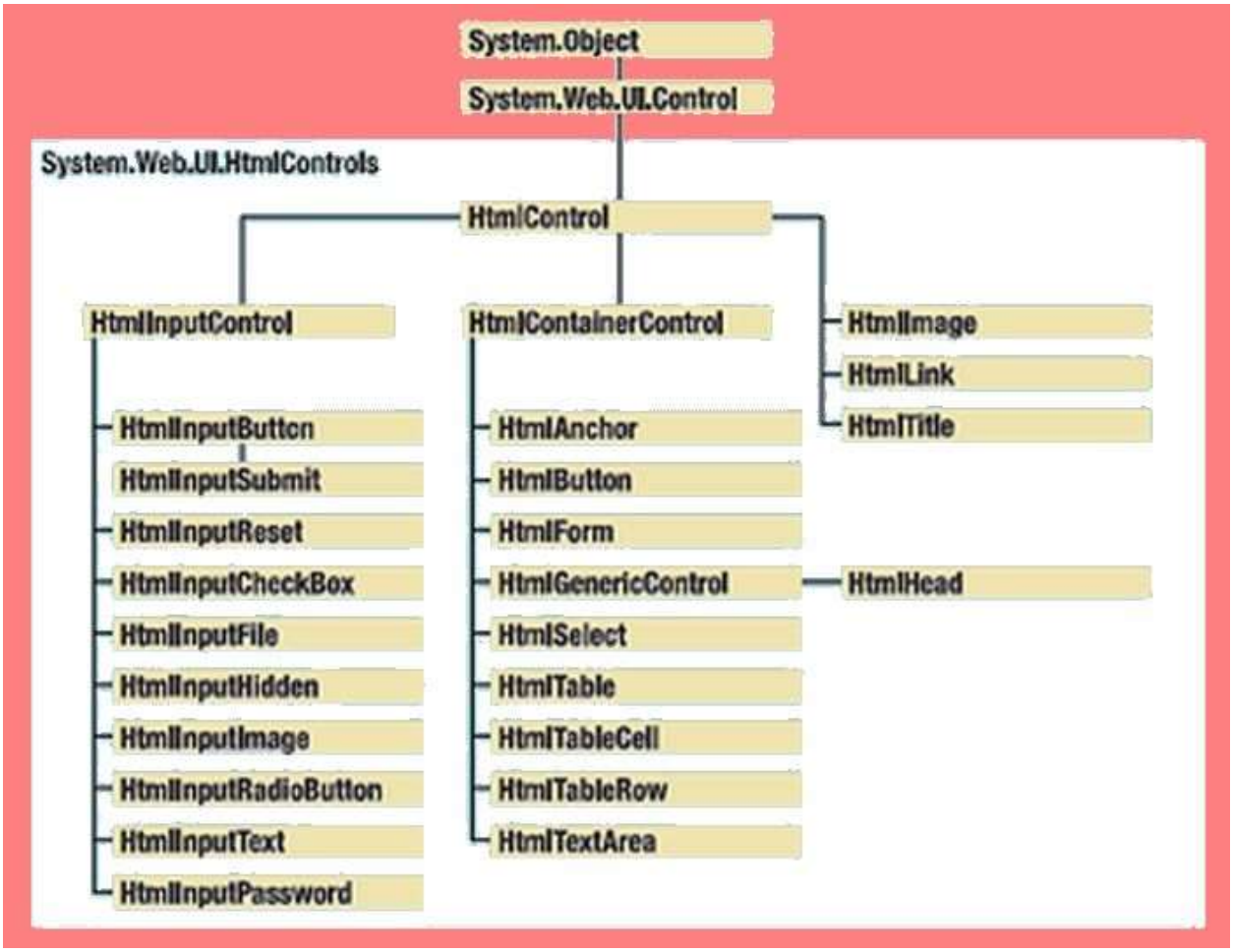
HasControls():

Returns boolean value indicating whether control has any child control.

Render():

Generates html from the control based on its current state.

HTML Server Controls



HtmlControl is the base class of all HTML server controls.

The HTML server controls are Hypertext Markup Language (HTML) elements that include a runat=server attribute. 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 System.Web.UI.HtmlControls.HtmlControl base class contains all of the common properties. HTML server controls derive from this class.

To use an HTML server control, use the following syntax (which uses the HtmlInputText control as an example):

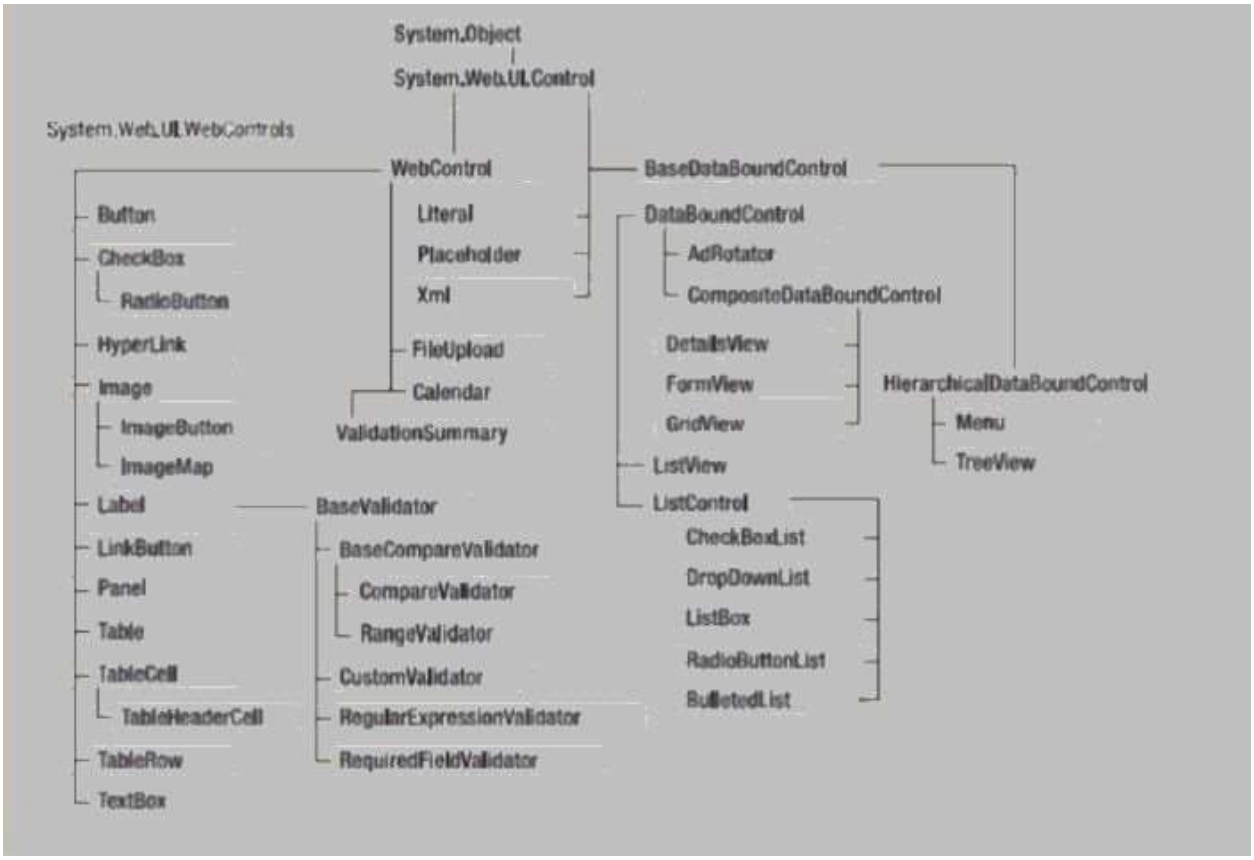
```
<input type="text" value="hello world" runat=server />
```

The following table describes the HTML server controls:

Control Name	HTML tag
HtmlHead	<head>element
HtmlInputButton	<input type=button|submit|reset>
HtmlInputCheckbox	<input type=checkbox>
HtmlInputFile	<input type = file>
HtmlInputHidden	<input type = hidden>
HtmlInputImage	<input type = image>
HtmlInputPassword	<input type = password>
HtmlInputRadioButton	<input type = radio>
HtmlInputReset	<input type = reset>
HtmlText	<input type = text|password>
HtmlImage	<img> element
HtmlLink	<link> element
HtmlAnchor	<a> element
HtmlButton	<button> element
HtmlButton	<button> element
HtmlForm	<form> element
HtmlTable	<table> element
HtmlTableCell	<td> and <th>
HtmlTableRow	<tr> element
HtmlTitle	<title> element
HtmlSelect	<select> element

Web Server Controls

All web controls are defined in a System.Web.UI.WebControls namespace and derive from WebControl base class. These web controls are able to generate not just single html tag but more complex output consists of several HTML tags and JavaScript code.
Figure shows the inheritance hierarchy of web controls.



Web server controls offer the following advantages:

- Make it easier for manufacturers and developers to build tools or applications that automatically generate the user interface.
- Simplify the process of creating interactive Web forms, which requires less knowledge of how HTML controls work and make the task of using them less prone to errors.

To use a Web server control, use the following syntax (which uses the TextBox control as an example):

```
<asp:textbox text="hello world" runat=server />
```

Web server controls can be divided into four categories:

- Basic Web Controls

- Validation Controls
- List Controls
- Rich Controls

Basic Web Controls

Basic Web controls provide the same functionality as their HTML server control counterparts. However, basic Web control include additional methods, events, and properties against which you can program.

Button Web Server Control, *CheckBox Web Server Control*, *HyperLink Web Server Control*, *Image Web Server Control*, *ImageButton Web Server Control*, *Label Web Server Control* are some of the basic web controls.

Validation Controls

Validation controls are used to validate the values that are entered into other controls of the page. Validation controls perform client-side validation, server-side validation, or both, depending on the capabilities of the browser in which the page is displayed. Validation controls offer the following advantages:

- You can associate one or more validation controls with each control that you want to validate.
- The validation is performed when the page form is submitted.
- You can specify programmatically whether validation should occur, which is useful if you want to provide a cancel button so that the user can exit without having to fill valid data in all of the fields.
- The validation controls automatically detect whether validation should be performed on the client side or the server side.

  Note A client-side validation catches errors before a postback operation is complete. Therefore, if you have combinations of client-side and server-side validation controls on a single page, the server-side validation will be preempted if a client-side validation fails.

In ASP.NET following validation controls exists –

- **RequireFieldValidator**: Checks value of the control is not empty when form is submitted.
- **RangeValidator**: Checks the value of the control is in specified range.
- **CompareValidator**: Checks the value of the control matches comparison against another control value or value.
- **RegularExpressionValidator**: Checks the value of the control matches specific RegularExpression.
- **CustomValidator**: Allows you to specify your custom validation logic (client side or server side)
- **ValidationSummary:** Shows Summary of Validation Messages generated by all Validation controls on web page or popup message.

All the Validation Controls are derived from BaseValidator class and are found in System.Web.UI.WebControls namespace.

List Controls

List controls are special Web server controls that support binding to collections. You can use list controls to display rows of data in a customized, templated format. All list controls expose DataSource and DataMember properties, which are used to bind to collections.

List controls can bind only to collections that support the IEnumerable, ICollection, or IListSourceinterfaces.

For example, a Microsoft Visual C# .NET sample page appears as follows:

```
<%@ Page Language="C#" %>
<script runat="server">
Public void Page_Load()
{
String[] myStringArray = new String[] {"one","two","three"};
rptr.DataSource = myStringArray;
rptr.DataBind();
}
</script>
<html>
<body>
<asp:repeater id=rptr runat="server">
<itemtemplate><%# Container.DataItem %><br></itemtemplate>
</asp:repeater>
</body>
```

```
</html>
```

A Microsoft Visual Basic .NET sample page appears as follows:

```
<%@ Page Language="vb" %>
<script runat="server">
public sub Page_Load()
   Dim myStringArray as String()
   myStringArray = new String() {"one","two","three"}
   rptr.DataSource = myStringArray
   rptr.DataBind()
end sub
</script>
<html>
<body>
<asp:repeater id=rptr runat="server">
   <itemtemplate><%# Container.DataItem %><br></itemtemplate>
</asp:repeater>
</body>
</html>
```

The output appears as follows:

one

two

three

ListBox Web Server Control, *CheckBoxList Web Server Control*, *RadioButtonList Web Server Control*, *Repeater Web Server Control*, *DataList Web Server Control*, *DataGrid Web Server Control*, *DropDownList Web Server Control* are some of the List Controls.

Rich Controls

In addition to the preceding controls, the ASP.NET page framework provides a few, task-specific controls called rich controls. Rich controls are built with multiple HTML elements and contain rich functionality. Examples of rich controls are the Calendar control and the AdRotator control.

AdRotator: This control is used to display banner ads from a set of images. You can display each image for a predefined schedule configured in xml file.

Calendar: This control is used to display date. User can move through months, days to select date.

User Controls

User controls behaves like miniature ASP.NET pages or web forms, which could be used by many other pages. These are derived from the System.Web.UI.UserControl class.

To convert a Web Form into a user control, follow these steps:

1. Remove all <html>,<head>, <body> and <form> tags.
2. If the @ Page directive appears in the page, change it to @ Control.
3. Include a className attribute in the @ Control directive so that the user control is typed strongly when you instantiate it.
4. Give the control a descriptive file name, and change the file extension from .aspx to .ascx.

Custom Controls

Custom control is a control that is not included in the .NET framework library and is instead created by a third-party software vendor or a user.

Custom control is a concept used while building both Windows Forms client and ASP.NET Web applications. Custom client controls are used in Windows Forms applications, while custom server controls are used in ASP.NET pages (Web forms). Using custom controls is easier in .NET than the earlier Windows versions due to simple programming techniques.

Custom control is a generic term that also includes user controls. User control in ASP.NET is created using ASP.NET code and is reused in other Web pages, whereas user control in the context of Windows Forms implies a composite control with a consistent user interface (UI) and behavior within or across applications.

STATE MANAGEMENT

No web application framework, no matter how advanced, can change the fact that HTTP is a stateless protocol. After every web request, the client disconnects from the server, and the ASP.NET engine discards the objects that were created for the page. This architecture ensures that web applications can scale up to serve thousands of simultaneous requests without running out of server memory. The drawback is that your code needs to use other techniques to store information between web requests and retrieve it when needed.
In this section, you'll see how to tackle this challenge by maintaining information on the server and on the client using a variety of techniques. You'll also learn how to transfer information from one web page to another.

State Management changes in ASP.NET 4

ASP.NET 4 adds a few refinements to its state management features:

Opt-in view state: ASP.NET 4 adds a ViewStateMode property that allows you to disable view state for a page but then selectively enable view state for those controls that absolutely require it. This opt-in model of view state is described in the "Selectively Disabling View State" section.
Session compression: ASP.NET 4 introduces a compression feature that reduces the size of data before it's sent to an out-of-process state provider. This feature is described in the "Compression" section.
Selectively enabling session state: ASP.NET 4 adds the HttpContext.SetSessionStateBehavior() method. You can create an HTTP module (as described in Chapter 5) that examines the current request and then calls SetSessionStateBehavior() to programmatically enable or disable session state. The idea here is to wring just a bit more performance out of your web application by disabling session state when it's not needed but still allowing it to work for some requests. However, this is a fairly specialized optimization technique that most developers won't use.
Partial session state: Session state now recognizes the concept of *partial* state storage and retrieval, which could theoretically allow you to pull just a single property out of a serialized object. As promising as this sounds, no current state providers support it, so you can't use this feature in your applications just yet. Microsoft may release session state providers that support this feature in future versions of ASP.NET or sooner—for example, with new products like Windows Server AppFabric (http://tinyurl.com/yhds97y).

ASP.NET includes a variety of options for state management. You choose the right option depending on the data you need to store, the length of time you want to store it, the scope of your data (whether it's limited to individual users or shared across multiple requests), and additional security and performance considerations. The different state management options in ASP.NET are complementary, which means you'll almost always use a combination of them in the same web application (and often the same page).

State Management Options Compared (Part 1)

	View State	**Query String**	**Custom Cookies**
Allowed data types	All serializable .NET data types.	A limited amount of string data.	String data.
Storage location	A hidden field in the current web page.	The browser's URL string.	The client's computer (in memory or a small text file, depending on its lifetime settings).
Lifetime	Retained permanently for postbacks to a single page.	Lost when the user enters a new URL or closes the browser. However, can be stored and can persist between visits.	Set by the programmer. It can be used in multiple pages and it persists between visits.
Scope	Limited to the current page.	Limited to the target page.	The whole ASP.NET application.
Security	Tamper-proof by default but easy to read. You can use the Page directive to enforce encryption.	Clearly visible and easy for the user to modify.	Insecure and can be modified by the user.
Performance Implications	Storing a large amount of information will slow transmission but will not affect server performance.	None, because the amount of data is trivial.	None, because the amount of data is trivial.
Typical use	Page-specific settings.	Sending a product ID from a catalog page to a details page.	Personalization preferences for a website.

State Management Options Compared (Part 2)

	Session State	**Application State**
Allowed data types	All serializable .NET data types. Nonserializable types are supported if you are using the default in-process state service.	All .NET data types.

Storage location	Server memory (by default), or a dedicated database, depending on the mode you choose.	Server memory.
Lifetime	Times out after a predefined period (usually 20 minutes but can be altered globally or programmatically).	The lifetime of the application (typically, until the server is rebooted).
Scope	The whole ASP.NET application.	The whole ASP.NET application. Unlike most other types of methods, application data is global to all users.
Security	Secure, because data is never transmitted to the client. However, subject to session hijacking if you don't use SSL.	Very secure, because data is stored on the server.
Performance Implications	Storing a large amount of information can slow down the server severely, especially if there are a large number of users at once, because each user will have a separate set of session data.	Storing a large amount of information can slow down the server, because this data will never time out and be removed.
Typical use	Store items in a shopping basket.	Storing any type of global data.

State Management Options Compared (Part 3)

	Profiles	**Caching**
Allowed data types	All serializable .NET data types.	All .NET data types. Nonserializable types are supported if you create a custom profile.
Storage location	A back-end database.	Server memory.
Lifetime	Permanent.	Depends on the expiration policy you set, but may possibly be released early if server memory becomes scarce.
Scope	The whole ASP.NET application. May also be	The same as application state (global

	accessed by other applications.	to all users and all pages).
Security	Fairly secure, because although data is never transmitted, it is stored without encryption in a database that could be compromised.	Very secure, because the cached data is stored on the server.
Performance implications	Large amounts of data can be stored easily, but there may be a nontrivial overhead in retrieving and writing the data for each request.	Storing a large amount of information may force out other, more useful cached information. However, ASP.NET has the ability to remove items early to ensure optimum performance.
Typical use	Store customer account information.	Storing data retrieved from a database.

View State

View state should be your first choice for storing information within the bounds of a single page. View state
is used natively by the ASP.NET web controls. It allows them to retain their properties between postbacks.
You can add your own data to the view state collection using a built-in page property called ViewState. The
type of information you can store includes simple data types and your own custom objects.
Like most types of state management in ASP.NET, view state relies on a dictionary collection, where each item is indexed with a unique string name. For example, consider this code:

```
ViewState["Counter"] = 1;
```

This places the value 1 (or rather, an integer that contains the value 1) into the ViewState collection and gives it the descriptive name Counter. If there is currently no item with the name Counter, a new item
will be added automatically. If there is already an item indexed under the name Counter, it will be replaced.
When retrieving a value, you use the key name. You also need to cast the retrieved value to the appropriate data type. This extra step is required because the ViewState collection casts all items to the base Object type, which allows it to handle any type of data.
Here's the code that retrieves the counter from view state and converts it to an integer:

```
int counter;
if (ViewState["Counter"] != null)
{
counter = (int)ViewState["Counter"];
}
```

If you attempt to look up a value that isn't present in the collection, you'll receive a NullReferenceException. To defend against this possibility, you should check for a null value before you attempt to retrieve and cast data that may not be present.

A View State Example

Another approach to saving data for the user, is the ViewState. As described elsewhere in this tutorial, the ViewState allows ASP.NET to repopulate form fields on each postback to the server, making sure that a form is not automatically cleared when the user hits the submit button. All this happens automatically, unless you turn it off, but you can actually use the ViewState for your own purposes as well. Please keep in mind though, that while cookies and sessions can be accessed from all your pages on your website, ViewState values are not carried between pages. Here is a simple example of using the ViewState to carry values between postbacks:

```
<%@ 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 itself. 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!

Accessing View State

View state is ideal because it doesn't take up any memory on the server and doesn't impose any arbitrary usage limits (such as a time-out). So, what might force you to abandon view state for another type of state management? Here are three possible reasons:

• You need to store mission-critical data that the user cannot be allowed to tamper with. (An ingenious user could modify the view state information in a postback request.) In this case, consider session state. Alternatively, consider using the countermeasures described in the next section. They aren't bulletproof, but they will greatly increase the effort an attacker would need in order to read or modify view state data.

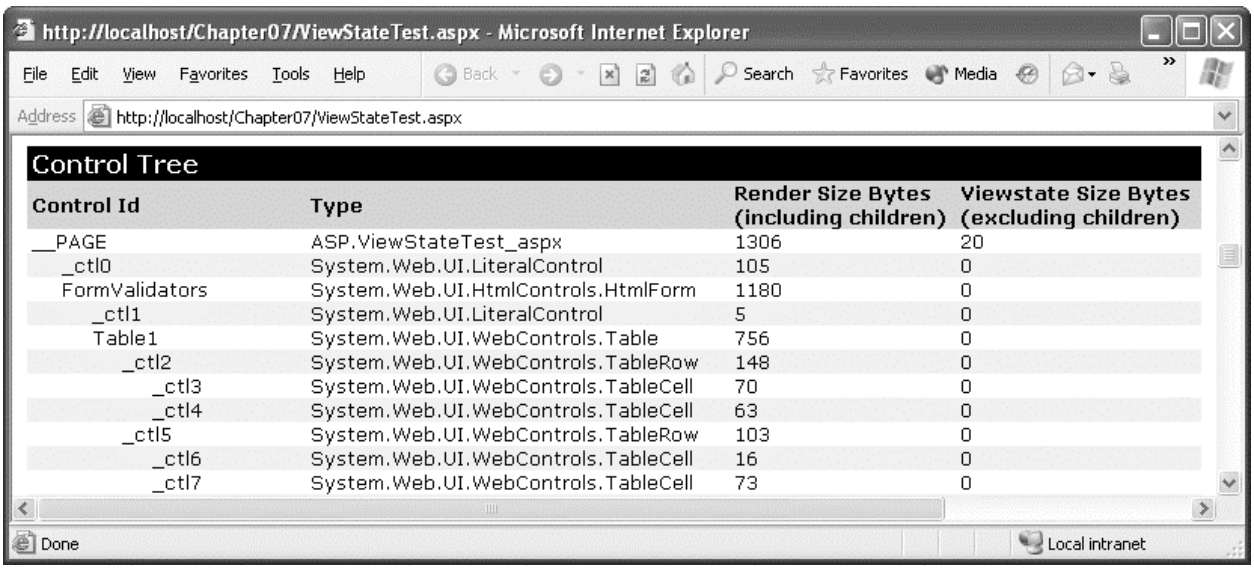
• You need to store information that will be used by multiple pages. In this case, consider session state, cookies, or the query string.

• You need to store an extremely large amount of information, and you don't want to slow down page transmission times. In this case, consider using a database, or possibly session state.

The amount of space used by view state depends on the number of controls, their complexity, and the amount of dynamic information. If you want to profile the view state usage of a page, just turn on tracing by adding the Trace attribute to the Page directive, as shown here:

<%@ Page Language="C#" Trace="true" ... %>

Look for the Control Tree section. Although it doesn't provide the total view state used by the page, it does indicate the view state used by each individual control in the Viewstate Size Bytes column (Figure below). Don't worry about the Render Size Bytes column, which simply reflects the size of the rendered HTML for the control.



Selectively Disabling View State

To improve the transmission times of your page, it's a good idea to eliminate view state when it's not needed. Although you can disable view state at the application and page level, it makes the most sense to
disable it on a per-control basis. You won't need view state for a control in three instances:

- The control never changes. For example, a button with static text doesn't need view state.
- The control is repopulated in every postback. For example, if you have a label that shows the current time, and you set the current time in the Page.Load event handler, it doesn't need view state.
- The control is an input control, and it changes only because of user actions. After each postback, ASP.NET will populate your input controls using the submitted form values. This means the text in a text box or the selection in a list box won't be lost, even if you don't use view state.

To turn off view state for a single control, set the EnableViewState property of the control to false. To turn off view state for an entire page and all its controls, set the EnableViewState property of the page to
false, or use the EnableViewState attribute in the Page directive, as shown here:

```
<%@ Page Language="C#" EnableViewState="false" ... %>
```

Even when you disable view state for the entire page, you'll still see the hidden view state tag with a small amount of information in the rendered HTML. That's because ASP.NET always stores the control hierarchy for the page at a minimum. There's no way to remove this last little fragment of data.
You can turn view state off for all the web pages in your application by setting the enableViewState attribute of the <pages> element in the web.config file, as shown here:

```
<configuration>
<system.web>
<pages enableViewState="false" />
...
</system.web>
</configuration>
```

Now, you'll need to set the EnableViewState attribute of the Page directive to true if you want to switch on view state for a particular page.
Finally, it's possible to switch of view state for a page (either through the Page directive or through the web.config file) but selectively override that setting by explicitly enabling view state for a particular control. This technique, which is new in ASP.NET 4, is popular with developers who are obsessed with paring down the view state of their pages to the smallest size possible. It allows you to switch on view state only when it's absolutely necessary—for example, with a data editing control such as the GridView (which uses view state to keep track of the currently selected item, among other details).
To use this approach, you need to use another property, called ViewStateMode. Like EnableViewState, the ViewStateMode property applies to all controls and page and can be set in a control tag or through an attribute in the page directive. ViewStateMode takes one of three values:

Enabled: View state will work, provided the EnableViewState property allows it.
Disabled: View state will not work for this control, although it may be allowed for child controls.
Inherit: This control will use the ViewStateMode property of its container. This is the default value.

To use opt-in state management, you set ViewStateMode of the page to Disabled. This turns off view state for the top-level page. By default, all the controls inside the page will have a ViewStateMode of Inherit, which means they also disable themselves.

```
<%@ Page Language="C#" ViewStateMode="Disabled" ... %>
```

Note that you do *not* set EnableViewState to false—if you do, ASP.NET completely shuts down view state for the page, and no control can opt in.
Now, to opt in for a particular control in the page, you simply set ViewStateMode to Enabled:

```
<asp:Label ViewStateMode="Enabled" ... />
```

This model is a bit awkward, but it's useful when view state size is an issue. The only drawback is that you need to remember to explicitly enable view state on controls that have dynamic values you want to persist or on controls that use view state for part of their functionality.

The Query String

One common approach is to pass information using a query string in the URL. You will commonly find this approach 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 "organic+gardening" string.

The advantage of the query string is that it's lightweight and doesn't exert any kind of burden on the server. Unlike cross-page posting, the query string can easily transport the same information from page to page. It has some limitations, however:

- Information is limited to simple strings, which must contain URL-legal characters.
- Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
- The enterprising user might decide to modify the query string and supply new values, which your program won't expect and can't protect against.
- Many browsers impose a limit on the length of a URL (usually from 1 to 2 KB). For that reason, you can't place a large amount of information in the query string and still be assured of compatibility with most browsers.

Adding information to the query string is still a useful technique. It's particularly well suited in database applications where you present the user with a list of items corresponding to records in a

database, like products. The user can then select an item and be forwarded to another page with detailed
information about the selected item. One easy way to implement this design is to have the first page send the item ID to the second page. The second page then looks that item up in the database and displays the detailed information. You'll notice this technique in e-commerce sites such as Amazon.com.

Using the Query String

To store information in the query string, you need to place it there yourself. Unfortunately, there is no collection-based way to do this. Typically, this means using a special HyperLink control, or you can use a Response.Redirect() statement like the one shown here:

```
// Go to newpage.aspx. Submit a single query string argument
// named recordID and set to 10.
int recordID = 10;
Response.Redirect("newpage.aspx?recordID=" + recordID.ToString());
```

You can send multiple parameters as long as you separate them with an ampersand (&), as shown here:

```
// Go to newpage.aspx. Submit two query string arguments:
// recordID (10) and mode (full).
Response.Redirect("newpage.aspx?recordID=10&mode=full");
```

The receiving page has an easier time working with the query string. It can receive the values from the QueryString dictionary collection exposed by the built-in Request object, as shown here:

```
string ID = Request.QueryString["recordID"];
```

If the query string doesn't contain the recordID parameter, or if the query string contains the recordID parameter but doesn't supply a value, the ID string will be set to null.

Note that information is always retrieved as a string, which can then be converted to another simple data type. Values in the QueryString collection are indexed by the variable name.

Cookies

Custom cookies provide another way you can store information for later use. Cookies are small files that are created on the client's hard drive (or, if they're temporary, in the web browser's memory). One advantage of cookies is that they work transparently without the user being aware that information needs to be stored. They also can be easily used by any page in your application and even retained between visits, which allows for truly long-term storage. They suffer from some of the same drawbacks that affect query strings. Namely, they're limited to simple string information, and they're easily accessible and readable if the user finds and opens the corresponding file. These factors make them a poor choice for complex or private information or large amounts of data.

Some users disable cookies on their browsers, which will cause problems for web applications that require them. However, cookies are widely adopted because so many sites use them.
Cookies are fairly easy to use. Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection. The important trick to remember is that you retrieve cookies from the Request object, and you set cookies using the Response object.

To set a cookie, just create a new System.Net.HttpCookie object. You can then fill it with string information (using the familiar dictionary pattern) and attach it to the current web response, as follows:

```
// 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 (which is stored with the temporary Internet files on the user's hard drive), you can set an expiration date, as shown here:

```
// This cookie lives for one year.
cookie.Expires = DateTime.Now.AddYears(1);
```

Cookies are retrieved by cookie name using the Request.Cookies collection, as shown here:

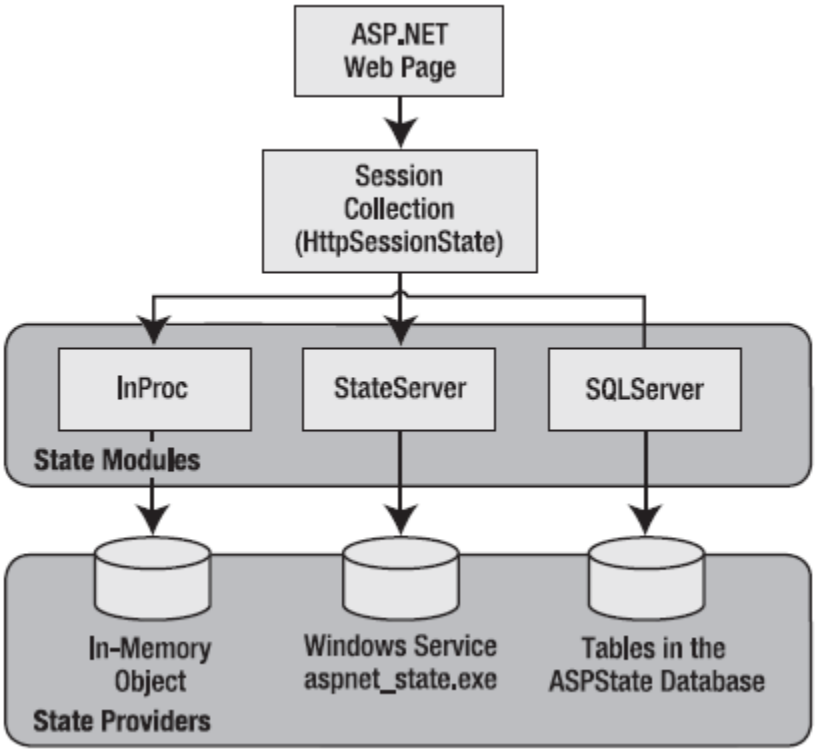
```
HttpCookie cookie = Request.Cookies["Preferences"];

// Check to see whether a cookie was found with this name.
// This is a good precaution to take,
// because the user could disable cookies,
// in which case the cookie would not exist.
string language;
if (cookie != null)
{
  language = cookie["LanguagePref"];
}
```

The only way to remove a cookie is by replacing it with a cookie that has an expiration date that has already passed. The following code demonstrates this technique:

```
HttpCookie cookie = new HttpCookie("LanguagePref");
cookie.Expires = DateTime.Now.AddDays(-1);
Response.Cookies.Add(cookie);
```

Session Architecture

Session management is not part of the HTTP standard. As a result, ASP.NET needs to do some extra work to track session information and bind it to the appropriate response.
ASP.NET tracks each session using a unique 120-bit identifier. ASP.NET uses a proprietary algorithm to generate this value, thereby guaranteeing (statistically speaking) that the number is unique and that it's random enough so a malicious user can't reverse-engineer or guess what session ID a given client will be using. This ID is the only piece of 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 serialized data from the state server, converts it to live objects, and places these objects into a special collection so they can be accessed in code. This process takes place automatically.



Session state is another example of ASP.NET's pluggable architecture. A state provider is any class that implements the *IHttpSessionState* interface, which means you can customize how session state works simply by building (or purchasing) a new .NET component. ASP.NET includes three prebuilt state providers, which allow you to store information in process, in a separate service, or in a SQL Server database.

For session state to work, the client needs to present the appropriate session ID with each request. The final ingredient in the puzzle is how the session ID is tracked from one request to the next. You can accomplish this in two ways:

Using cookies: In this case, the session ID is transmitted in a special cookie (named ASP.NET_SessionId), which ASP.NET creates automatically when the session collection is used. This is the default, and it's also the same approach that was used in earlier versions of ASP.

Using modified URLs: In this case, the session ID is transmitted in a specially modified (or "munged") URL. This allows you to create applications that use session state with clients that don't support cookies.

Using Session State

You can interact with session state using the System.Web.SessionState.HttpSessionState class, which is provided in an ASP.NET web page as the built-in Session object. The syntax for adding items to the collection and retrieving them is basically the same as for adding items to the view state of a page.

For example, you might store a DataSet in session memory like this:

```
Session["ProductsDataSet"] = dsProducts;
```

You can then retrieve it with an appropriate conversion operation:

```
dsProducts = (DataSet)Session["ProductsDataSet"];
```

Session state is global to your entire application for the current user. Session state can be lost in several ways:

- If the user closes and restarts the browser.
- If the user accesses the same page through a different browser window, although the session will still exist if a web page is accessed through the original browser window. Browsers differ on how they handle this situation.
- If the session times out because of inactivity. By default, a session times out after 20 idle minutes.
- If the programmer ends the session by calling `Session.Abandon()`.

In the first two cases, the session actually remains in memory on the server, because the web server has no idea that the client has closed the browser or changed windows. The session will linger in memory, remaining inaccessible, until it eventually expires.

In addition, session state will be lost when the application domain is re-created. This process happens transparently when you update your web application or change a configuration setting. The application domain may also be recycled periodically to ensure application health. If this behaviour is causing a problem, you can store session state information out of process, as described in the next section. With out-of-process state storage, the session information is retained even when the application domain is shut down.

Mode

The mode session state settings allow you to configure what session state provider is used to store session state information between requests. The following sections explain your options.

Off

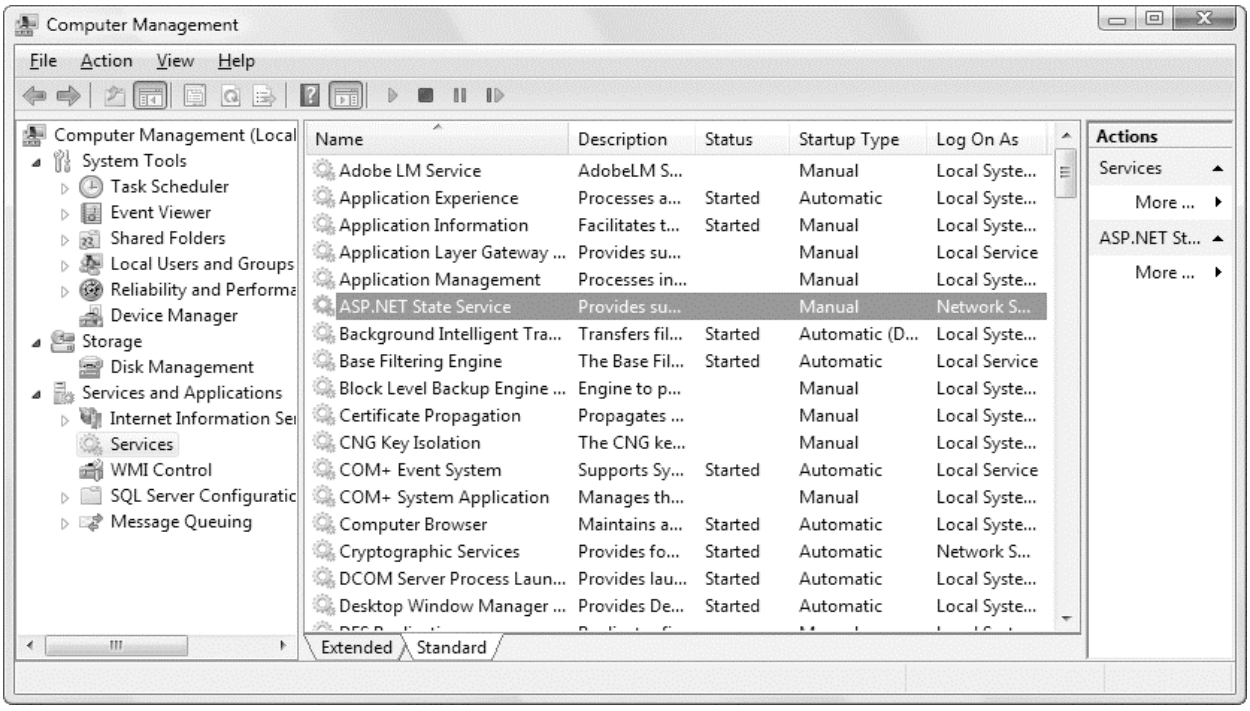
This setting disables session state management for every page in the application. This can provide a slight performance improvement for websites that are not using session state.

InProc

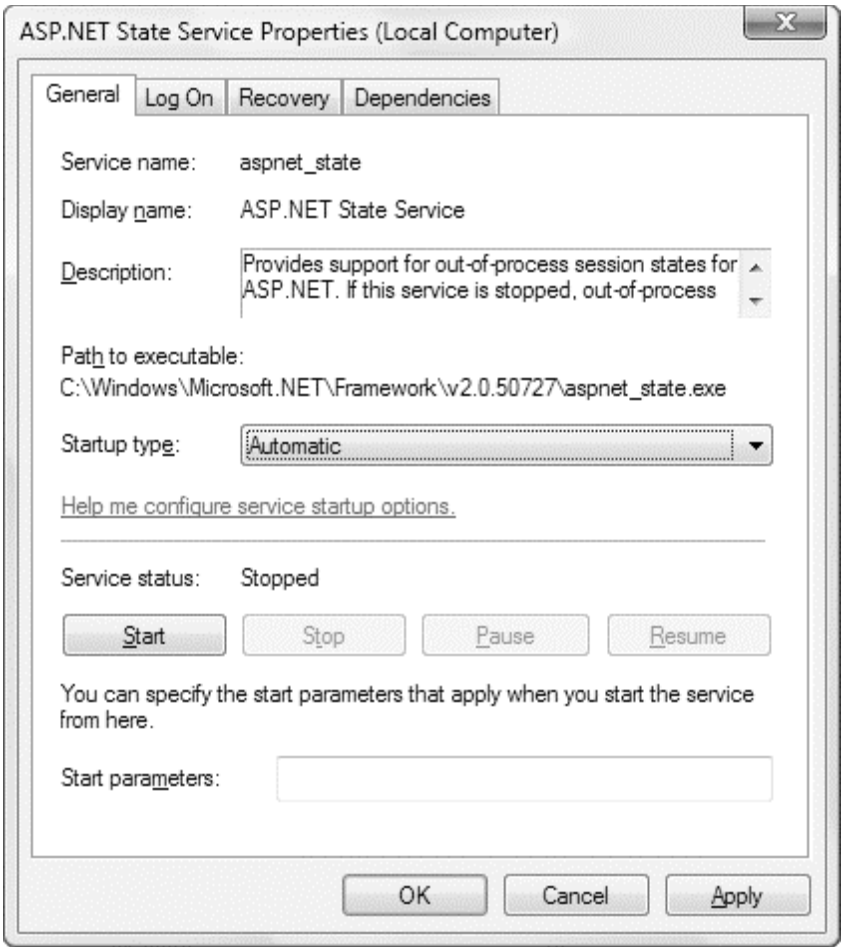
InProc is similar to how session state was stored in classic ASP. It instructs ASP.NET to store information in the current application domain. This provides the best performance but the least durability. If you restart your server, the state information will be lost.
InProc is the default option, and it makes sense for most small websites.

State Server

With this setting, ASP.NET will use a separate Windows service for state management. Even if you run this service on the same web server, it will be loaded outside the main ASP.NET process, which gives it a basic level of protection if the ASP.NET process needs to be restarted. The cost is the increased time delay imposed when state information is transferred between two processes. If you frequently access and change state information, this can make for a fairly unwelcome slowdown.
When using the StateServer setting, you need to specify a value for the stateConnectionString setting. This string identifies the TCP/IP address of the computer that is running the StateServer service and its port number (which is defined by ASP.NET and doesn't usually need to be changed). This allows you to host the StateServer on another computer. If you don't change this setting, the local server will be
used (set as address 127.0.0.1).
Of course, before your application can use the service, you need to start it. The easiest way to do this is to use the Microsoft Management Console. Select Start ➤ Programs ➤ Administrative Tools ➤ Computer Management (you can also access the Administrative Tools group through the Control Panel). Then, in the Computer Management tool, find the Services and Applications ➤ Services node. Find the service called ASP.NET State Service in the list, as shown in Figure below.



Once you find the service in the list, you can manually start and stop it by right-clicking it. Generally, you'll want to configure Windows to automatically start the service. Right-click it, select Properties, and modify the Startup Type setting to Automatic, as shown in Figure below. Then click Start to start it immediately.



SQL Server

This setting instructs ASP.NET to use a SQL Server database to store session information, as identified by the sqlConnectionString attribute. This is the most resilient state store but also the slowest by far. To use
this method of state management, you'll need to have a server with SQL Server installed.
When setting the sqlConnectionString, you follow the same sort of pattern you use with ADO.NET data access. Generally, you'll need to specify a data source (the server
address) and a user ID and password, unless you're using SQL integrated security.
In addition, you need to install the special stored procedures and temporary session databases. These stored procedures take care of storing and retrieving the session information. ASP.NET includes a command-line tool that does the work for you automatically, called aspnet_regsql.exe. It's found in the c:\Windows\Microsoft.NET\Framework\[Version] directory. The easiest way to run aspnet_regsql.exe is to start by launching the Visual Studio command prompt (open the Start menu and choose Programs ➤ Visual Studio 2010 ➤ Visual Studio Tools ➤ Visual Studio 2010 Command Prompt). You can then type in an aspnet_regsql.exe command, no matter what directory you're in.

Here's a command that creates the session storage database on the current computer, using the default database name ASPState:

```
aspnet_regsql.exe -S localhost -E -ssadd
```

This command uses the alias localhost, which tells aspnet_regsql.exe to connect to the database server on the current computer. You can replace this detail with the computer name of your database server.
Once you've created your session state database, you need to tell ASP.NET to use it by modifying the `<sessionState>` section of the web.config file. If you're using a database named ASPState to store your session information (which is the default), you don't need to supply the database name. Instead, you simply need to indicate the location of the server and the type of authentication that ASP.NET should use to connect to it, as shown here:

```
<sessionState mode="SQLServer"
sqlConnectionString="data source=localhost;Integrated Security=SSPI"
... />
```

Additionally, the state tables will be removed every time you restart SQL Server, no matter what the session time-out. That's because the state tables are created in the tempdb database, which is a temporary storage area. If this isn't the behavior you want, you can tell the aspnet_regsql.exe tool to install permanent state tables in the ASPState database. To do this, you use the -sstype p (for persisted) parameter. Here's the revised command line:

```
aspnet_regsql.exe -S localhost -E -ssadd -sstype p
```

Now session records will remain in the database, even if you restart SQL Server.
Your final option is to use aspnet_regsql.exe to create the state tables in a different database (not ASPState). To do so, you use the -sstype c (for custom) parameter, and then supply the database name with the -d parameter, as shown here:

```
aspnet_regsql.exe -S localhost -E -ssadd -sstype c -d MyCustomStateDb
```

When you use this approach, you'll create permanent session tables, so their records will remain even when SQL Server is restarted.

Cookieless

You can set the cookieless setting to one of the values defined by the HttpCookieMode enumeration. You can also set the name that's used for the cookie with the cookieName attribute. If you don't, the default value cookie name is ASP.NET_SessionId.

Here's an example that forces cookieless mode (which is useful for testing):
`<sessionState cookieless="UseUri" ... />`
In cookieless mode, the session ID will automatically be inserted into the URL. When ASP.NET receives a request, it will remove the ID, retrieve the session collection, and forward the request to the appropriate directory. A munged URL is shown here:

```
http://localhost/WebApplication/(amfvyc55evojk455cffbq355)/Page1.aspx
```

Because the session ID is inserted in the current URL, relative links also automatically gain the session ID. In other words, if the user is currently stationed on Page1.aspx and clicks a relative link to Page2.aspx, the relative link includes the current session ID as part of the URL. The same is true if you call Response.Redirect() with a relative URL, as shown here:

```
Response.Redirect("Page2.aspx");
```

The only real limitation of cookieless state is that you cannot use absolute links, because they will not contain the session ID. For example, this statement causes the user to lose all session information:

```
Response.Redirect("http://localhost/WebApplication/Page2.aspx");
```

By default, ASP.NET allows you to reuse a session identifier. For example, if you make a request and your query string contains an expired session, ASP.NET creates a new session and uses that session ID. The problem is that a session ID might inadvertently appear in a public place—such as in a results page in a search engine. This could lead to multiple users accessing the server with the same session identifier and then all joining the same session with the same shared data.
To avoid this potential security risk, it's recommended that you include the optional regenerateExpiredSessionId attribute and set it to true whenever you use cookieless sessions. This way, a new session ID will be issued if a user connects with an expired session ID. The only drawback is that this process also forces the current page to lose all view state and form data, because ASP.NET performs a redirect to make sure the browser has a new session identifier.

Timeout

Another important session state setting in the web.config file is the timeout. This specifies the number of minutes that ASP.NET will wait, without receiving a request, before it abandons the session.

```
<sessionState timeout="20" ... />
```

This setting represents one of the most important compromises of session state. A difference of minutes can have a dramatic effect on the load of your server and the performance of your application.
Ideally, you will choose a time frame that is short enough to allow the server to reclaim valuable memory after a client stops using the application but long enough to allow a client to pause and continue a session without losing it.
You can also programmatically change the session time-out in code. For example, if you know a session contains an unusually large amount of information, you may need to limit the amount of time the session can be stored. You would then warn the user and change the timeout property. Here's a sample line of code that changes the time-out to ten minutes:

```
Session.Timeout = 10;
```

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 types of objects, retains information on the server, and uses the same dictionary-based syntax. A common example with application state is a global counter that tracks how many times an operation has been performed by all of the web application's clients.
For example, you could create a `global.asax` event handler that tracks how many sessions have been created or how many requests have been received into the application. Or you can use similar logic in the `Page.Load` event handler to track how many times a given page has been requested by various clients. Here's an example of the latter:

```
protected void Page_Load(Object sender, EventArgs e)
{
 int count = 0;
 if (Application["HitCounterForOrderPage"] != null)
  count = (int)Application["HitCounterForOrderPage"];

 count++;
 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 until the application domain refreshes itself (because of automatic process-recycling settings or an update to one of the pages or components in the application).
Application state isn't often used, because it's generally inefficient. In the previous example, the counter would probably not keep an accurate count, particularly in times of heavy traffic. For example, if two clients requested the page at the same time, you could have a sequence of events like this:

1. User A retrieves the current count (432).
2. User B retrieves the current count (432).
3. User A sets the current count to 433.
4. User B sets the current count to 433.

In other words, one request isn't counted because two clients access the counter at the same time.
To prevent this problem, you need to use the Lock() and UnLock() methods, which explicitly allow only one client to access the Application state collection at a time, as follows:

```
protected void Page_Load(Object sender, EventArgs e)
{
 // Acquire exclusive access.
 Application.Lock();
 int count = 0;
```

```
 if (Application["HitCounterForOrderPage"] != null)
   count = (int)Application["HitCounterForOrderPage"];

 count++;
 Application["HitCounterForOrderPage"] = count;
 // Release exclusive access.
 Application.UnLock();
 lblCounter.Text = count.ToString();
}
```

The Web.config file

Every web application inherits the settings from the machine.config file and the root web.config file. In addition, you can apply settings to individual web applications. For example, you might want to set a specific method for authentication, a type of debugging, a default language, or custom error pages. To do so, you supply a web.config file in the root virtual directory of your web application. To further configure individual subdirectories in your web application, you can place additional web.config files in these folders.

It's important to understand that the web.config file in a web application can't override all the settings in the machine.config file. Certain settings, such as the process model settings, can't be changed on a per-application basis. Other settings are application-specific. That means you can set them in the web.config file that's in the root virtual directory of your website, but you can't set them using a web.config file that's in a subdirectory.
The entire content of an ASP.NET configuration file is nested in a root <configuration> element.

This element contains a `<system.web>` element, which is used for ASP.NET settings. Inside the `<system.web>` element are separate elements for each aspect of configuration. Along with `<system.web>` are the `<appSettings>` element%, which you can use to store custom settings, and the `<connectionStrings>` element, which you can use to store connection strings to databases that you use or that other ASP.NET features rely on.

Here is the absolute simplest web.config file, which is what you get when you create a blank ASP.NET website in Visual Studio:

```
<?xml version="1.0"?>
<configuration>
<system.web>
<compilation debug="true" targetFramework="4.0" />
</system.web>
</configuration>
```

The <system.web> section is the heart of ASP.NET configuration. Inside it are all the elements that configure ASP.NET features.

```
<?xml version="1.0"?>
<configuration>
 <appSettings />
 <connectionStrings />
 <system.web>
  <!-- ASP.NET configuration sections go here. -->
 </system.web>
 <system.webServer />
</configuration>
```

<system.web>

The <system.web> element contains all the ASP.NET-specific configuration settings. These settings configure various aspects of your web application and enable services such as security, state management, and tracing. The schema of the <system.web> section is fixed—in other words, you can't change the structure or add your own custom elements here. However, you can include as few or as many configuration sections as you want.

Some basic configuration sections.

Element	Description
authentication	This element configures your authorization system—in other words, it determines how you will verify a client's identity when the client requests a page.
authorization	This element controls which clients have access to the resources within the web application or current directory.
compilation	This element identifies the version of .NET that your web application is targeting (through the targetFramework attribute) and whether you want to generate debug symbols in .pdb files (through the debug attribute), so you can debug your application with a tool like Visual Studio. The compilation element can also contain the <assemblies> element, which lists additional assemblies that your web application uses. These assemblies are then made available to your code (as long as they can be found in the Bin directory or the GAC).
customErrors	This element allows you to set specific redirect URLs that should be used when specific (or default) errors occur. For example, this element could be used to redirect the user to a friendly replacement for the dreaded 404 (page not found) error. But although this setting still works with Visual Studio's built-in test web server, it's effectively been replaced by the <httpErrors> section in IIS 7.x.
membership	This element allows you to configure ASP.NET's membership feature, which manages user account information and provides a high-level API for security-related tasks such as user login and password resetting.
pages	This element defines default page settings (most of which you can override with the Page directive).
profile	This element allows you to configure ASP.NET's profile feature, which automatically stores and retrieves user-specific information (usually, profile settings). Typically, profile data is serialized to a database.

roleManager	This element allows you to configure ASP.NET's role-based security feature, which provides a way to store role information and a high-level API for role-based authorization.
sessionState	This element configures the various options for maintaining session state for the application, such as whether to maintain it at all and where to maintain it (SQL, a separate Windows service, and so on).
trace	This element configures tracing, an ASP.NET feature that lets you display diagnostic information in the page (or collect it for viewing separately).

<system.webserver>

This section contains settings that affect to the web server. You use the <handlers> element inside this section to register custom HTTP handlers. You use the <modules> section to register HTTP modules.

<appSettings>

You add custom settings to a web.config file in a special element called <appSettings>. Here's where the <appSettings> section fits into the web.config file:

```
<?xml version="1.0"?>
<configuration>
 <appSettings>
 <!-- Custom application data goes here. -->
 </appSettings>
<system.web>...</system.web>
</configuration>
```

Custom settings are entered using an <add> element that identifies a unique variable name (the key) and the variable contents (the value). The following example adds two new custom configuration settings:

```
<?xml version="1.0" ?>
<configuration>
<appSettings>
<add key="websiteName" value="My New Website"/>
<add key="welcomeMessage" value="Welcome to my new Website, friend!"/>
</appSettings>
<system.web>...</system.web>
</configuration>
```

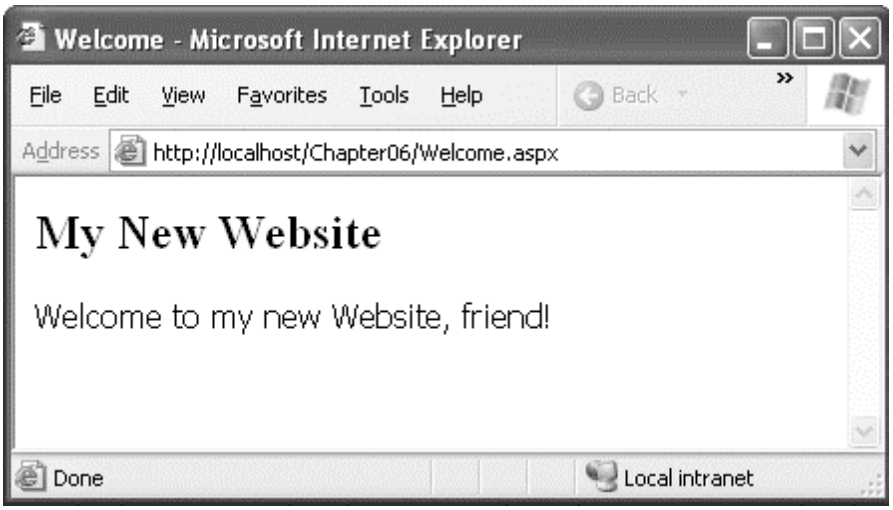
Once you've added this information, .NET makes it extremely easy to retrieve it in your web-page code. You simply need to use the WebConfigurationSettings class from the System.Web.Configuration namespace. It exposes a static property called AppSettings, which contains a dynamically built collection of available application settings for the current directory. For example, if the ASP.NET page class

referencing the AppSettings collection is at a location such as `http://localhost/MyApp/MyDirectory/MySubDirectory`, it is possible that the AppSettings collection
contains settings from three different web.config files. The AppSettings collection makes that hierarchy seamless to the page that's using it.
To use the WebConfigurationSettings class, it helps to first import the System.Web.Configuration namespace so you can refer to the class without needing to use the long fully qualified name, as shown here:

```
using System.Web.Configuration;
```

Next, you simply need to retrieve the value by name. The following example fills two labels using the custom application information:

```
protected void Page_Load(object sender, EventArgs e)
{
 lblSiteName.Text =
 WebConfigurationManager.AppSettings["websiteName"];
 lblWelcome.Text =   WebConfigurationManager.AppSettings["welcomeMessage"];
}
```



An error won't occur if you try to retrieve a value that doesn't exist. If you suspect this could be a problem, make sure to test for a null reference before retrieving a value.

<connectionStrings>

This section allows you to define database connection strings that will be used elsewhere in your application. Seeing as connection strings need to be reused exactly to support connection pooling and may need to be modified without recompiling the web application, it makes perfect sense to store them in the web.config file.

You can add as many connection strings as you want. For each one, you need to specify the ADO.NET provider.
Here's an example that defines a single connection string:

```
<configuration>
  <connectionStrings>
    <add name="NorthwindConnection"
         connectionString=
  "Data Source=localhost;Integrated Security=SSPI;Initial Catalog=Northwind;"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>...</system.web>
</configuration>
```

You can retrieve connection strings in your code using the static WebConfigurationManager.ConnectionStrings property:

```
string connectionString =
WebConfigurationManager.ConnectionStrings["NorthwindConnection"].Value;
```

The ConnectionStrings collection includes the connection strings that are defined directly in your web.config file and any that are defined in higher-level configuration files (namely, the root web.config file and the machine.config file). That means you'll automatically get a connection string named LocalSqlServer that points to a local instance of SQL Server Express (which is the scaled-down version of SQL Server that's included with Visual Studio). The connection string looks like this:

```
Data Source=.\SQLEXPRESS;Integrated Security=SSPI;
AttachDBFilename=|DataDirectory|aspnetdb.mdf; User Instance=true
```

<The global.asax Application file>

The global.asax file allows you to write event handlers that react to global events. Users cannot request the global.asax file directly. Instead, the global.asax file executes its code automatically in response to certain application events. The global.asax file provides a similar service to the global.asa file in classic ASP applications.
You write the code in a global.asax file in a similar way to a web form. The difference is that the global.asax doesn't contain any HTML or ASP.NET tags. Instead, it contains methods with specific, predefined names. For example, the following global.asax file reacts to the HttpApplication.EndRequest event, which happens just before the page is sent to the user:

```
<%@ Application Language="C#" %>
<script language="C#" runat="server">
protected void Application_OnEndRequest()

{
Response.Write("<hr />This page was served at " +
DateTime.Now.ToString());
}
</script>
```

Although it's not indicated in the global.asax file, every global.asax file defines the methods for a single class—the application class. The application class derives from HttpApplication, and as a result your code has access to all its public and protected members. This example uses the Response object, which is provided as a built-in property of the HttpApplication class, just like it's a built-in property of the Page class.
In the preceding example, the Application_OnEndRequest() event handler writes a footer at the bottom of the page with the date and time that the page was created. Because it reacts to the HttpApplication.EndRequest event, this method executes every time a page is requested, after all the event-handling code in that page has finished.
As with web forms, you can also separate the content of the global.asax file into two files, one that declares the file and another that contains the code. However, because there's no design surface for global.asax files, the division isn't required. Visual Studio doesn't give you the option to create a global.asax file with a separate code-behind class.

The global.asax file is optional, but a web application can have no more than one global.asax file, and it must reside in the root directory of the application, not in a subdirectory. To add a global.asax file to a project, select Website ➤ Add New Item (or Project ➤ Add New Item if you're using the Visual Studio web project model) and choose the Global Application Class template. (This option doesn't appear if you already have a global.asax file in your project.) When Visual Studio adds a global.asax file, it includes empty event handlers for the most commonly used application events. You simply need to insert your code in the appropriate method.

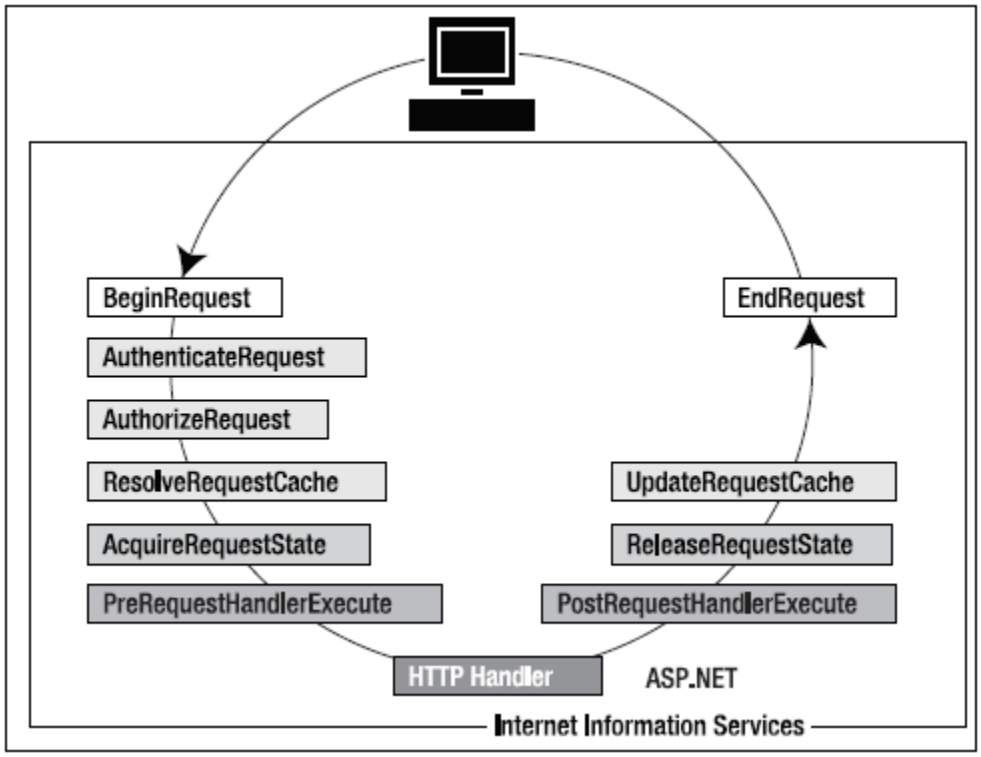
Application Events

You can handle two types of events in the global.asax file:
• Events that always occur for every request. These include request-related and response-related events.
• Events that occur only under certain conditions.

The required events unfold in this order:

1. **Application_BeginRequest():** This method is called at the start of every request.
2. **Application_AuthenticateRequest():** This method is called just before authentication is performed. This is a jumping-off point for creating your own authentication logic.
3. **Application_AuthorizeRequest():** After the user is authenticated (identified), it's time to determine the user's permissions. You can use this method to assign special privileges.
4. **Application_ResolveRequestCache():** This method is commonly used in conjunction with output caching. With output caching, the rendered HTML of a web form is reused, without executing any of your code. However, this event handler still runs.
5. At this point, the request is handed off to the appropriate handler. For example, for a web form request, this is the point when the page is compiled (if necessary) and instantiated.
6. **Application_AcquireRequestState():** This method is called just before session specific information is retrieved for the client and used to populate the Session collection.
7. **Application_PreRequestHandlerExecute():** This method is called before the appropriate HTTP handler executes the request.
8. At this point, the appropriate handler executes the request. For example, if it's a web form request, the event-handling code for the page is executed, and the page is rendered to HTML.
9. **Application_PostRequestHandlerExecute():** This method is called just after the request is handled.
10. **Application_ReleaseRequestState():** This method is called when the session specific information is about to be serialized from the Session collection so that it's available for the next request.
11. **Application_UpdateRequestCache():** This method is called just before information is added to the output cache. For example, if you've enabled output caching for a web page, ASP.NET will insert the rendered HTML for the page into the cache at this point.
12. **Application_EndRequest():** This method is called at the end of the request, just before the objects are released and reclaimed. It's a suitable point for cleanup code.

Below figure shows the process of handling a single request.



Some events don't fire with every request:

Application_Start(): This method is invoked when the application first starts up and the application domain is created. This event handler is a useful place to provide application-wide initialization code. For example, at this point you might load and cache data that will not change throughout the lifetime of an application, such as navigation trees, static product catalogs, and so on.

Session_Start(): This method is invoked each time a new session begins. This is often used to initialize user-specific information. Chapter 6 discusses sessions with state management.

Application_Error(): This method is invoked whenever an unhandled exception occurs in the application.

Session_End(): This method is invoked whenever the user's session ends. A session ends when your code explicitly releases it or when it times out after there have been no more requests received within a given timeout period (typically 20 minutes). This method is typically used to clean up any related data. However, this method is only called if you are using in-process session state storage (the InProc mode, not the StateServer or SQLServer modes).

Application_End(): This method is invoked just before an application ends. The end of an application can occur because IIS is being restarted or because the application is transitioning to a new application domain in response to updated files or the process recycling settings.

Application_Disposed(): This method is invoked some time after the application has been shut down and the .NET garbage collector is about to reclaim the memory it occupies. This point is too late to perform critical cleanup, but you can use it as a last-ditch failsafe to verify that critical resources are released.

Application events are commonly used to perform application initialization, cleanup, usage logging, profiling, and troubleshooting. However, don't assume that your application will need to use global application events. Many ASP.NET applications don't use the global.asax file at all.

STATE MANAGEMENT

No web application framework, no matter how advanced, can change the fact that HTTP is a stateless protocol. After every web request, the client disconnects from the server, and the ASP.NET engine discards the objects that were created for the page. This architecture ensures that web applications can scale up to serve thousands of simultaneous requests without running out of server memory. The
drawback is that your code needs to use other techniques to store information between web requests and retrieve it when needed.
In this section, you'll see how to tackle this challenge by maintaining information on the server and on the client using a variety of techniques. You'll also learn how to transfer information from one web page to another.

State Management changes in ASP.NET 4

ASP.NET 4 adds a few refinements to its state management features:

Opt-in view state: ASP.NET 4 adds a ViewStateMode property that allows you to disable view state for a page but then selectively enable view state for those controls that absolutely require it. This opt-in model of view state is described in the "Selectively Disabling View State" section.
Session compression: ASP.NET 4 introduces a compression feature that reduces the size of data before it's sent to an out-of-process state provider. This feature is described in the "Compression" section.
Selectively enabling session state: ASP.NET 4 adds the HttpContext.SetSessionStateBehavior() method. You can create an HTTP module (as described in Chapter 5) that examines the current request and then calls SetSessionStateBehavior() to programmatically enable or disable session state. The idea here is to wring just a bit more performance out of your web application by disabling session state when it's not needed but still allowing it to work for some requests. However, this is a fairly specialized optimization technique that most developers won't use.
Partial session state: Session state now recognizes the concept of *partial* state storage and retrieval, which could theoretically allow you to pull just a single property out of a serialized object. As promising as this sounds, no current state providers support it, so you can't use this feature in your applications just yet. Microsoft may release session state providers that support this feature in future versions of ASP.NET or sooner—for example, with new products like Windows Server AppFabric (http://tinyurl.com/yhds97y).

ASP.NET includes a variety of options for state management. You choose the right option depending on
the data you need to store, the length of time you want to store it, the scope of your data (whether it's
limited to individual users or shared across multiple requests), and additional security and performance
considerations. The different state management options in ASP.NET are complementary, which means
you'll almost always use a combination of them in the same web application (and often the same page).

State Management Options Compared (Part 1)

	View State	**Query String**	**Custom Cookies**
Allowed data types	All serializable .NET data types.	A limited amount of string data.	String data.
Storage location	A hidden field in the current web page.	The browser's URL string.	The client's computer (in memory or a small text file, depending on its lifetime settings).
Lifetime	Retained permanently for postbacks to a single page.	Lost when the user enters a new URL or closes the browser. However, can be stored and can persist between visits.	Set by the programmer. It can be used in multiple pages and it persists between visits.
Scope	Limited to the current page.	Limited to the target page.	The whole ASP.NET application.
Security	Tamper-proof by default but easy to read. You can use the Page directive to enforce encryption.	Clearly visible and easy for the user to modify.	Insecure and can be modified by the user.
Performance Implications	Storing a large amount of information will slow transmission but will not affect server performance.	None, because the amount of data is trivial.	None, because the amount of data is trivial.
Typical use	Page-specific settings.	Sending a product ID from a catalog page to a details page.	Personalization preferences for a website.

State Management Options Compared (Part 2)

	Session State	**Application State**
Allowed data types	All serializable .NET data types. Nonserializable types are supported if you are using the default in-process	All .NET data types.

	state service.	
Storage location	Server memory (by default), or a dedicated database, depending on the mode you choose.	Server memory.
Lifetime	Times out after a predefined period (usually 20 minutes but can be altered globally or programmatically).	The lifetime of the application (typically, until the server is rebooted).
Scope	The whole ASP.NET application.	The whole ASP.NET application. Unlike most other types of methods, application data is global to all users.
Security	Secure, because data is never transmitted to the client. However, subject to session hijacking if you don't use SSL.	Very secure, because data is stored on the server.
Performance Implications	Storing a large amount of information can slow down the server severely, especially if there are a large number of users at once, because each user will have a separate set of session data.	Storing a large amount of information can slow down the server, because this data will never time out and be removed.
Typical use	Store items in a shopping basket.	Storing any type of global data.

State Management Options Compared (Part 3)

	Profiles	**Caching**
Allowed data types	All serializable .NET data types.	All .NET data types. Nonserializable types are supported if you create a custom profile.
Storage location	A back-end database.	Server memory.
Lifetime	Permanent.	Depends on the expiration policy you set, but may possibly be released early if server memory becomes scarce.
Scope	The whole ASP.NET application. May also be accessed by other applications.	The same as application state (global to all users and all pages).

Security	Fairly secure, because although data is never transmitted, it is stored without encryption in a database that could be compromised.	Very secure, because the cached data is stored on the server.
Performance implications	Large amounts of data can be stored easily, but there may be a nontrivial overhead in retrieving and writing the data for each request.	Storing a large amount of information may force out other, more useful cached information. However, ASP.NET has the ability to remove items early to ensure optimum performance.
Typical use	Store customer account information.	Storing data retrieved from a database.

View State

View state should be your first choice for storing information within the bounds of a single page. View state
is used natively by the ASP.NET web controls. It allows them to retain their properties between postbacks.
You can add your own data to the view state collection using a built-in page property called ViewState. The
type of information you can store includes simple data types and your own custom objects.
Like most types of state management in ASP.NET, view state relies on a dictionary collection, where each item is indexed with a unique string name. For example, consider this code:

```
ViewState["Counter"] = 1;
```

This places the value 1 (or rather, an integer that contains the value 1) into the ViewState collection and gives it the descriptive name Counter. If there is currently no item with the name Counter, a new item
will be added automatically. If there is already an item indexed under the name Counter, it will be replaced.
When retrieving a value, you use the key name. You also need to cast the retrieved value to the appropriate data type. This extra step is required because the ViewState collection casts all items to the
base Object type, which allows it to handle any type of data.
Here's the code that retrieves the counter from view state and converts it to an integer:

```
int counter;
if (ViewState["Counter"] != null)
{
counter = (int)ViewState["Counter"];
}
```

If you attempt to look up a value that isn't present in the collection, you'll receive a NullReferenceException. To defend against this possibility, you should check for a null value before you
attempt to retrieve and cast data that may not be present.

A View State Example

Another approach to saving data for the user, is the ViewState. As described elsewhere in this tutorial, the ViewState allows ASP.NET to repopulate form fields on each postback to the server, making sure that a form is not automatically cleared when the user hits the submit button. All this happens automatically, unless you turn it off, but you can actually use the ViewState for your own purposes as well. Please keep in mind though, that while cookies and sessions can be accessed from all your pages on your website, ViewState values are not carried between pages. Here is a simple example of using the ViewState to carry values between postbacks:

```
<%@ 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 itself. 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!

Accessing View State

View state is ideal because it doesn't take up any memory on the server and doesn't impose any arbitrary usage limits (such as a time-out). So, what might force you to abandon view state for another type of state management? Here are three possible reasons:

• You need to store mission-critical data that the user cannot be allowed to tamper with. (An ingenious user could modify the view state information in a postback request.) In this case, consider session state. Alternatively, consider using the countermeasures described in the next section. They aren't bulletproof, but they will greatly increase the effort an attacker would need in order to read or modify view state data.

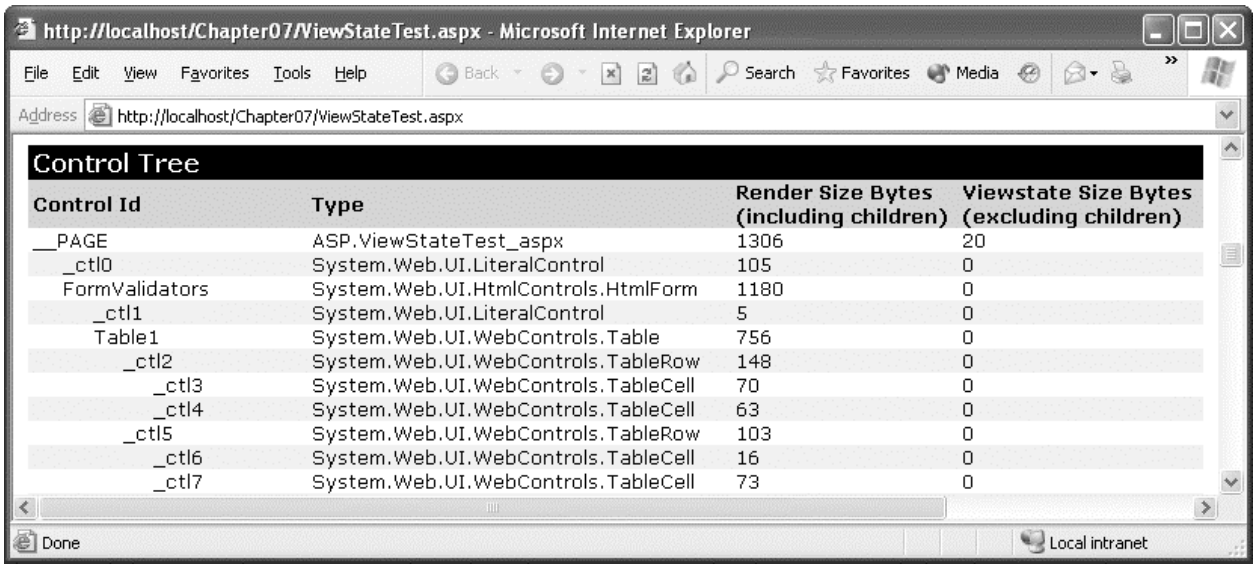
• You need to store information that will be used by multiple pages. In this case, consider session state, cookies, or the query string.

• You need to store an extremely large amount of information, and you don't want to slow down page transmission times. In this case, consider using a database, or possibly session state.

The amount of space used by view state depends on the number of controls, their complexity, and the amount of dynamic information. If you want to profile the view state usage of a page, just turn on
tracing by adding the Trace attribute to the Page directive, as shown here:

```
<%@ Page Language="C#" Trace="true" ... %>
```

Look for the Control Tree section. Although it doesn't provide the total view state used by the page, it does indicate the view state used by each individual control in the Viewstate Size Bytes column (Figure below). Don't worry about the Render Size Bytes column, which simply reflects the size of the
rendered HTML for the control.



Selectively Disabling View State

To improve the transmission times of your page, it's a good idea to eliminate view state when it's not
needed. Although you can disable view state at the application and page level, it makes the most sense to
disable it on a per-control basis. You won't need view state for a control in three instances:

- The control never changes. For example, a button with static text doesn't need view state.

- The control is repopulated in every postback. For example, if you have a label that shows the current time, and you set the current time in the Page.Load event handler, it doesn't need view state.
- The control is an input control, and it changes only because of user actions. After each postback, ASP.NET will populate your input controls using the submitted form values. This means the text in a text box or the selection in a list box won't be lost, even if you don't use view state.

To turn off view state for a single control, set the EnableViewState property of the control to false. To
turn off view state for an entire page and all its controls, set the EnableViewState property of the page to
false, or use the EnableViewState attribute in the Page directive, as shown here:

```
<%@ Page Language="C#" EnableViewState="false" ... %>
```

Even when you disable view state for the entire page, you'll still see the hidden view state tag with a small amount of information in the rendered HTML. That's because ASP.NET always stores the control
hierarchy for the page at a minimum. There's no way to remove this last little fragment of data.
You can turn view state off for all the web pages in your application by setting the enableViewState attribute of the <pages> element in the web.config file, as shown here:

```
<configuration>
<system.web>
<pages enableViewState="false" />
...
</system.web>
</configuration>
```

Now, you'll need to set the EnableViewState attribute of the Page directive to true if you want to switch on view state for a particular page.
Finally, it's possible to switch of view state for a page (either through the Page directive or through the web.config file) but selectively override that setting by explicitly enabling view state for a particular
control. This technique, which is new in ASP.NET 4, is popular with developers who are obsessed with
paring down the view state of their pages to the smallest size possible. It allows you to switch on view
state only when it's absolutely necessary—for example, with a data editing control such as the GridView
(which uses view state to keep track of the currently selected item, among other details).
To use this approach, you need to use another property, called ViewStateMode. Like EnableViewState, the ViewStateMode property applies to all controls and page and can be set in a control tag or through an attribute in the page directive. ViewStateMode takes one of three values:

Enabled: View state will work, provided the EnableViewState property allows it.
Disabled: View state will not work for this control, although it may be allowed for child controls.
Inherit: This control will use the ViewStateMode property of its container. This is the default value.

To use opt-in state management, you set ViewStateMode of the page to Disabled. This turns off view state for the top-level page. By default, all the controls inside the page will have a ViewStateMode of Inherit, which means they also disable themselves.

```
<%@ Page Language="C#" ViewStateMode="Disabled" ... %>
```

Note that you do *not* set EnableViewState to false—if you do, ASP.NET completely shuts down view state for the page, and no control can opt in.
Now, to opt in for a particular control in the page, you simply set ViewStateMode to Enabled:

```
<asp:Label ViewStateMode="Enabled" ... />
```

This model is a bit awkward, but it's useful when view state size is an issue. The only drawback is that you need to remember to explicitly enable view state on controls that have dynamic values you want to persist or on controls that use view state for part of their functionality.

The Query String

One common approach is to pass information using a query string in the URL. You will commonly find
this approach 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 "organic+gardening" string.

The advantage of the query string is that it's lightweight and doesn't exert any kind of burden on the server. Unlike cross-page posting, the query string can easily transport the same information from page
to page. It has some limitations, however:

- Information is limited to simple strings, which must contain URL-legal characters.
- Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
- The enterprising user might decide to modify the query string and supply new values, which your program won't expect and can't protect against.
- Many browsers impose a limit on the length of a URL (usually from 1 to 2 KB). For that reason, you can't place a large amount of information in the query string and still be assured of compatibility with most browsers.

Adding information to the query string is still a useful technique. It's particularly well suited in database applications where you present the user with a list of items corresponding to records in a database, like products. The user can then select an item and be forwarded to another page with detailed
information about the selected item. One easy way to implement this design is to have the first page send the item ID to the second page. The second page then looks that item up in the database and displays the detailed information. You'll notice this technique in e-commerce sites such as Amazon.com.

Using the Query String

To store information in the query string, you need to place it there yourself. Unfortunately, there is no collection-based way to do this. Typically, this means using a special HyperLink control, or you can use a Response.Redirect() statement like the one shown here:

```
// Go to newpage.aspx. Submit a single query string argument
// named recordID and set to 10.
int recordID = 10;
Response.Redirect("newpage.aspx?recordID=" + recordID.ToString());
```

You can send multiple parameters as long as you separate them with an ampersand (&), as shown here:

```
// Go to newpage.aspx. Submit two query string arguments:
// recordID (10) and mode (full).
Response.Redirect("newpage.aspx?recordID=10&mode=full");
```

The receiving page has an easier time working with the query string. It can receive the values from the QueryString dictionary collection exposed by the built-in Request object, as shown here:

```
string ID = Request.QueryString["recordID"];
```

If the query string doesn't contain the recordID parameter, or if the query string contains the recordID parameter but doesn't supply a value, the ID string will be set to null.

Note that information is always retrieved as a string, which can then be converted to another simple data type. Values in the QueryString collection are indexed by the variable name.

Cookies

Custom cookies provide another way you can store information for later use. Cookies are small files that are created on the client's hard drive (or, if they're temporary, in the web browser's memory). One advantage of cookies is that they work transparently without the user being aware that information needs to be stored. They also can be easily used by any page in your application and even retained between visits, which allows for truly long-term storage. They suffer from some of the same drawbacks that affect query strings. Namely, they're limited to simple string information, and they're easily accessible and readable if the user finds and opens the corresponding file. These factors make them a poor choice for complex or private information or large amounts of data.

Some users disable cookies on their browsers, which will cause problems for web applications that require them. However, cookies are widely adopted because so many sites use them.

Cookies are fairly easy to use. Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection. The important trick to remember is that you retrieve cookies from the Request object, and you set cookies using the Response object.

To set a cookie, just create a new System.Net.HttpCookie object. You can then fill it with string

information (using the familiar dictionary pattern) and attach it to the current web response, as follows:

```
// 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 (which is stored with the temporary Internet files on the user's
hard drive), you can set an expiration date, as shown here:

```
// This cookie lives for one year.
cookie.Expires = DateTime.Now.AddYears(1);
```

Cookies are retrieved by cookie name using the Request.Cookies collection, as shown here:

```
HttpCookie cookie = Request.Cookies["Preferences"];

// Check to see whether a cookie was found with this name.
// This is a good precaution to take,
// because the user could disable cookies,
// in which case the cookie would not exist.
string language;
if (cookie != null)
{
  language = cookie["LanguagePref"];
}
```

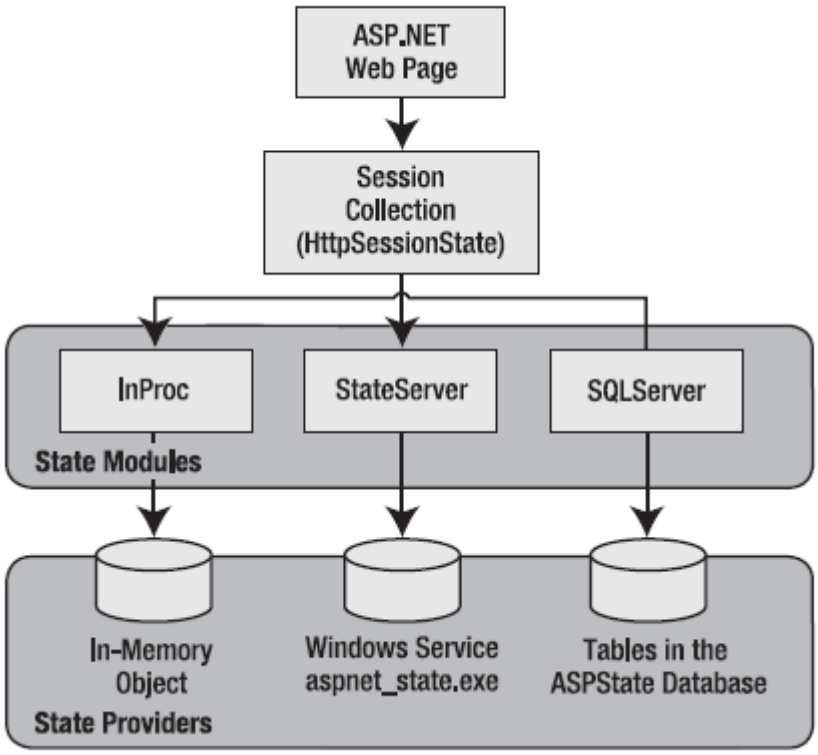
The only way to remove a cookie is by replacing it with a cookie that has an expiration date that has already passed. The following code demonstrates this technique:

```
HttpCookie cookie = new HttpCookie("LanguagePref");
cookie.Expires = DateTime.Now.AddDays(-1);
Response.Cookies.Add(cookie);
```

Session Architecture

Session management is not part of the HTTP standard. As a result, ASP.NET needs to do some extra work
to track session information and bind it to the appropriate response.
ASP.NET tracks each session using a unique 120-bit identifier. ASP.NET uses a proprietary algorithm to generate this value, thereby guaranteeing (statistically speaking) that the number is unique and that
it's random enough so a malicious user can't reverse-engineer or guess what session ID a given client

will be using. This ID is the only piece of 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 serialized data from the state server, converts it to live objects, and places these objects into a special collection so they can be accessed in code. This process takes place automatically.



Session state is another example of ASP.NET's pluggable architecture. A state provider is any class that implements the *IHttpSessionState* interface, which means you can customize how session state works simply by building (or purchasing) a new .NET component. ASP.NET includes three prebuilt state
providers, which allow you to store information in process, in a separate service, or in a SQL Server database.

For session state to work, the client needs to present the appropriate session ID with each request. The final ingredient in the puzzle is how the session ID is tracked from one request to the next. You can
accomplish this in two ways:

Using cookies: In this case, the session ID is transmitted in a special cookie (named ASP.NET_SessionId), which ASP.NET creates automatically when the session collection is used. This is the default, and it's also the same approach that was used in earlier versions of ASP.

Using modified URLs: In this case, the session ID is transmitted in a specially modified (or "munged") URL. This allows you to create applications that use session state with clients that don't

support cookies.

Using Session State

You can interact with session state using the System.Web.SessionState.HttpSessionState class, which is
provided in an ASP.NET web page as the built-in Session object. The syntax for adding items to the collection and retrieving them is basically the same as for adding items to the view state of a page.

For example, you might store a DataSet in session memory like this:

```
Session["ProductsDataSet"] = dsProducts;
```

You can then retrieve it with an appropriate conversion operation:

```
dsProducts = (DataSet)Session["ProductsDataSet"];
```

Session state is global to your entire application for the current user. Session state can be lost in several ways:

- If the user closes and restarts the browser.
- If the user accesses the same page through a different browser window, although the session will still exist if a web page is accessed through the original browser window. Browsers differ on how they handle this situation.
- If the session times out because of inactivity. By default, a session times out after 20 idle minutes.
- If the programmer ends the session by calling `Session.Abandon()`.

In the first two cases, the session actually remains in memory on the server, because the web server has no idea that the client has closed the browser or changed windows. The session will linger in memory, remaining inaccessible, until it eventually expires.

In addition, session state will be lost when the application domain is re-created. This process happens transparently when you update your web application or change a configuration setting. The application domain may also be recycled periodically to ensure application health. If this behaviour is causing a problem, you can store session state information out of process, as described in the next section. With out-of-process state storage, the session information is retained
even when the application domain is shut down.

Mode

The mode session state settings allow you to configure what session state provider is used to store session state information between requests. The following sections explain your options.

Off

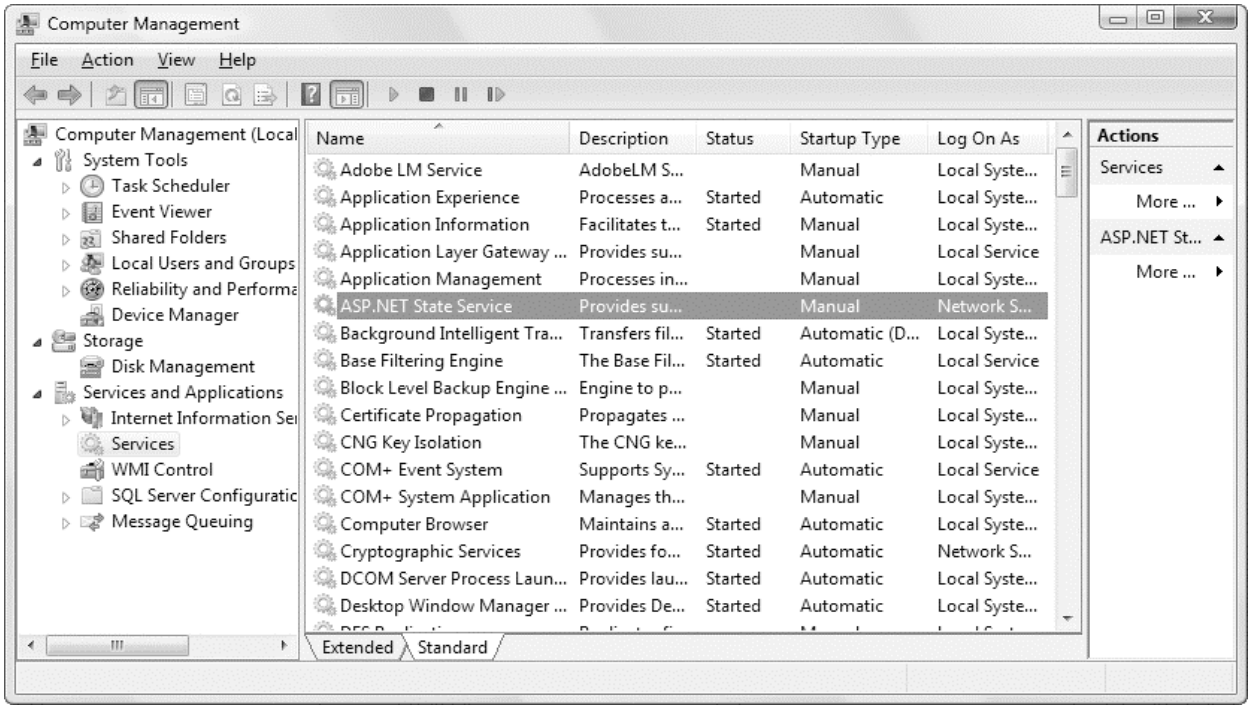
This setting disables session state management for every page in the application. This can provide a slight performance improvement for websites that are not using session state.

InProc

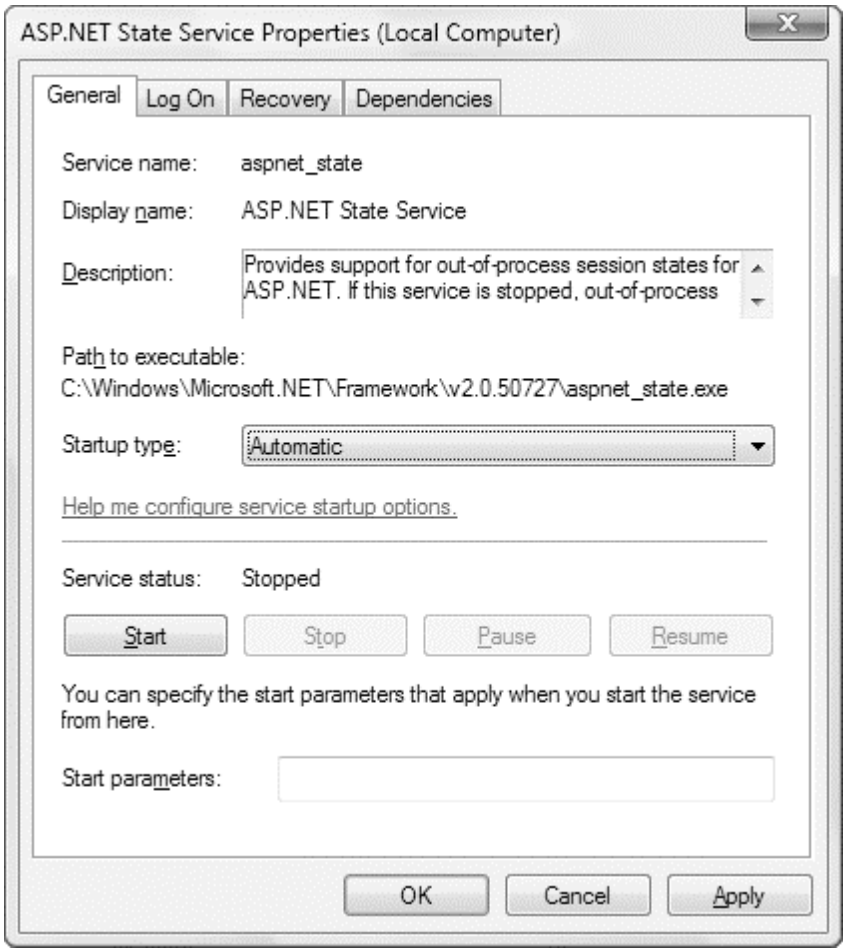
InProc is similar to how session state was stored in classic ASP. It instructs ASP.NET to store information
in the current application domain. This provides the best performance but the least durability. If you restart your server, the state information will be lost.
InProc is the default option, and it makes sense for most small websites.

State Server

With this setting, ASP.NET will use a separate Windows service for state management. Even if you run
this service on the same web server, it will be loaded outside the main ASP.NET process, which gives it a
basic level of protection if the ASP.NET process needs to be restarted. The cost is the increased time delay imposed when state information is transferred between two processes. If you frequently access
and change state information, this can make for a fairly unwelcome slowdown.
When using the StateServer setting, you need to specify a value for the stateConnectionString setting. This string identifies the TCP/IP address of the computer that is running the StateServer service
and its port number (which is defined by ASP.NET and doesn't usually need to be changed). This allows
you to host the StateServer on another computer. If you don't change this setting, the local server will be
used (set as address 127.0.0.1).
Of course, before your application can use the service, you need to start it. The easiest way to do this
is to use the Microsoft Management Console. Select Start ➤ Programs ➤ Administrative Tools ➤ Computer Management (you can also access the Administrative Tools group through the Control Panel).
Then, in the Computer Management tool, find the Services and Applications ➤ Services node. Find the service called ASP.NET State Service in the list, as shown in Figure below.



Once you find the service in the list, you can manually start and stop it by right-clicking it. Generally, you'll want to configure Windows to automatically start the service. Right-click it, select Properties, and modify the Startup Type setting to Automatic, as shown in Figure below. Then click Start to start it immediately.



SQL Server

This setting instructs ASP.NET to use a SQL Server database to store session information, as identified by
the sqlConnectionString attribute. This is the most resilient state store but also the slowest by far. To use
this method of state management, you'll need to have a server with SQL Server installed.
When setting the sqlConnectionString, you follow the same sort of pattern you use with ADO.NET data access. Generally, you'll need to specify a data source (the server
address) and a user ID and password, unless you're using SQL integrated security.
In addition, you need to install the special stored procedures and temporary session databases. These stored procedures take care of storing and retrieving the session information. ASP.NET includes a
command-line tool that does the work for you automatically, called aspnet_regsql.exe. It's found in the
c:\Windows\Microsoft.NET\Framework\[Version] directory. The easiest way to run aspnet_regsql.exe is
to start by launching the Visual Studio command prompt (open the Start menu and choose Programs ➤
Visual Studio 2010 ➤ Visual Studio Tools ➤ Visual Studio 2010 Command Prompt). You can then type in
an aspnet_regsql.exe command, no matter what directory you're in.

Here's a command that creates the session storage database on the current computer, using the default database name ASPState:

```
aspnet_regsql.exe -S localhost -E -ssadd
```

This command uses the alias localhost, which tells aspnet_regsql.exe to connect to the database server on the current computer. You can replace this detail with the computer name of your database
server.
Once you've created your session state database, you need to tell ASP.NET to use it by modifying the `<sessionState>` section of the web.config file. If you're using a database named ASPState to store your session information (which is the default), you don't need to supply the database name. Instead, you simply need to indicate the location of the server and the type of authentication that ASP.NET should use to connect to it, as shown here:

```
<sessionState mode="SQLServer"
sqlConnectionString="data source=localhost;Integrated Security=SSPI"
... />
```

Additionally, the state tables will be removed every time you restart SQL Server, no matter what the session time-out. That's because the state tables are created in the tempdb database, which is a temporary storage area. If this isn't the behavior you want, you can tell the aspnet_regsql.exe tool to install permanent state tables in the ASPState database. To do this, you use the -sstype p (for persisted)
parameter. Here's the revised command line:

```
aspnet_regsql.exe -S localhost -E -ssadd -sstype p
```

Now session records will remain in the database, even if you restart SQL Server.
Your final option is to use aspnet_regsql.exe to create the state tables in a different database (not

ASPState). To do so, you use the -sstype c (for custom) parameter, and then supply the database name
with the -d parameter, as shown here:

```
aspnet_regsql.exe -S localhost -E -ssadd -sstype c -d
MyCustomStateDb
```

When you use this approach, you'll create permanent session tables, so their records will remain even when SQL Server is restarted.

Cookieless

You can set the cookieless setting to one of the values defined by the HttpCookieMode enumeration. You can also set the name that's used for the cookie with the cookieName attribute. If you don't, the default value cookie name is ASP.NET_SessionId.

Here's an example that forces cookieless mode (which is useful for testing):

```
<sessionState cookieless="UseUri" ... />
```

In cookieless mode, the session ID will automatically be inserted into the URL. When ASP.NET receives a request, it will remove the ID, retrieve the session collection, and forward the request to the
appropriate directory. A munged URL is shown here:

```
http://localhost/WebApplication/(amfvyc55evojk455cffbq355)/Page1.asp
x
```

Because the session ID is inserted in the current URL, relative links also automatically gain the session ID. In other words, if the user is currently stationed on Page1.aspx and clicks a relative link to Page2.aspx, the relative link includes the current session ID as part of the URL. The same is true if you
call Response.Redirect() with a relative URL, as shown here:

```
Response.Redirect("Page2.aspx");
```

The only real limitation of cookieless state is that you cannot use absolute links, because they will not contain the session ID. For example, this statement causes the user to lose all session information:

```
Response.Redirect("http://localhost/WebApplication/Page2.aspx");
```

By default, ASP.NET allows you to reuse a session identifier. For example, if you make a request and your query string contains an expired session, ASP.NET creates a new session and uses that session ID.
The problem is that a session ID might inadvertently appear in a public place—such as in a results page
in a search engine. This could lead to multiple users accessing the server with the same session identifier
and then all joining the same session with the same shared data.
To avoid this potential security risk, it's recommended that you include the optional regenerateExpiredSessionId attribute and set it to true whenever you use cookieless sessions. This way, a new session ID will be issued if a user connects with an expired session ID. The only drawback is that

this process also forces the current page to lose all view state and form data, because ASP.NET performs
a redirect to make sure the browser has a new session identifier.

Timeout

Another important session state setting in the web.config file is the timeout. This specifies the number of minutes that ASP.NET will wait, without receiving a request, before it abandons the session.

```
<sessionState timeout="20" ... />
```

This setting represents one of the most important compromises of session state. A difference of minutes can have a dramatic effect on the load of your server and the performance of your application.
Ideally, you will choose a time frame that is short enough to allow the server to reclaim valuable memory after a client stops using the application but long enough to allow a client to pause and continue a session without losing it.
You can also programmatically change the session time-out in code. For example, if you know a session contains an unusually large amount of information, you may need to limit the amount of time
the session can be stored. You would then warn the user and change the timeout property. Here's a sample line of code that changes the time-out to ten minutes:

```
Session.Timeout = 10;
```

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 types of objects, retains information on the server, and uses the same dictionary-based syntax. A common example with application state is a global counter that tracks how many times an operation has been performed by all of the web application's clients.
For example, you could create a `global.asax` event handler that tracks how many sessions have been created or how many requests have been received into the application. Or you can use similar logic in the `Page.Load` event handler to track how many times a given page has been requested by various clients. Here's an example of the latter:

```
protected void Page_Load(Object sender, EventArgs e)
{
 int count = 0;
 if (Application["HitCounterForOrderPage"] != null)
  count = (int)Application["HitCounterForOrderPage"];

 count++;
 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 until the application domain refreshes itself (because of automatic process-recycling settings or an update to one of the pages or components in the application).
Application state isn't often used, because it's generally inefficient. In the previous example, the counter would probably not keep an accurate count, particularly in times of heavy traffic. For example, if two clients requested the page at the same time, you could have a sequence of events like this:

1. User A retrieves the current count (432).
2. User B retrieves the current count (432).
3. User A sets the current count to 433.
4. User B sets the current count to 433.

In other words, one request isn't counted because two clients access the counter at the same time. To prevent this problem, you need to use the Lock() and UnLock() methods, which explicitly allow only
one client to access the Application state collection at a time, as follows:

```
protected void Page_Load(Object sender, EventArgs e)
{
 // Acquire exclusive access.
 Application.Lock();
 int count = 0;

 if (Application["HitCounterForOrderPage"] != null)
```

```
   count = (int)Application["HitCounterForOrderPage"];

 count++;
 Application["HitCounterForOrderPage"] = count;
 // Release exclusive access.
 Application.UnLock();
 lblCounter.Text = count.ToString();
}
```

Programming ASP.NET Web Pages

ASP.NET is a framework that you can use to create dynamic web pages. A simple HTML web page is static; its content is determined by the fixed HTML markup that's in the page. Dynamic pages like those you create with ASP.NET web pages let you create the page content on the fly, by using the code.

Dynamic pages let you do all sorts of things. You can ask a user for input by using a form and then change what a page displays or how it looks. You can take a information from the user, save it in a database, and then list it later. You can send email from the site. You can interact with other services on the web and produce pages that integrate information from those sources.

Types and Variables

There are two kinds of types in C#: *value types* and *reference types*. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable. With value types, the variables each have their own copy of the data, and it is not possible for operations on one to affect the other (except in the case of ref and out parameter variables).

C#'s value types are further divided into *simple types*, *enum types*, *struct types*, and *nullable value types*. C#'s reference types are further divided into *class types*, *interface types*, *array types*, and *delegate types*.

The following provides an overview of C#'s type system.

Value types

Simple types

Signed integral: sbyte, short, int, long

Unsigned integral: byte, ushort, uint, ulong

Unicode characters: char

IEEE binary floating-point: float, double

High-precision decimal floating-point: decimal

Boolean: bool

Enum types

User-defined types of the form enum E {...}

Struct types

User-defined types of the form struct S {...}

Nullable value types

Extensions of all other value types with a null value

Reference types

Class types

Ultimate base class of all other types: object

Unicode strings: string

User-defined types of the form class C {...}

Interface types

User-defined types of the form interface I {...}

Array types

Single- and multi-dimensional, for example, int[] and int[,]

Delegate types

User-defined types of the form delegate int D(...)

C#'s bool type is used to represent Boolean values—values that are either true or false.

Character and string processing in C# uses Unicode encoding. The char type represents a UTF-16 code unit, and the string type represents a sequence of UTF-16 code units.

C# programs use type declarations to create new types. A type declaration specifies the name and the members of the new type. Five of C#'s categories of types are user-definable: class types, struct types, interface types, enum types, and delegate types.

A class type defines a data structure that contains data members (fields) and function members (methods, properties, and others). Class types support single inheritance and polymorphism, mechanisms whereby derived classes can extend and specialize base classes.

A struct type is similar to a class type in that it represents a structure with data members and function members. However, unlike classes, structs are value types and do not typically require heap allocation. Struct types do not support user-specified inheritance, and all struct types implicitly inherit from type object.

An interface type defines a contract as a named set of public function members. A class or struct that implements an interface must provide implementations of the interface's function members. An interface may inherit from multiple base interfaces, and a class or struct may implement multiple interfaces.

A delegate type represents references to methods with a particular parameter list and return type. Delegates make it possible to treat methods as entities that can be assigned to variables and passed as parameters. Delegates are analogous to function types provided by functional languages. They are also similar to the concept of function pointers found in some other languages, but unlike function pointers, delegates are object-oriented and type-safe.

The class, struct, interface and delegate types all support generics, whereby they can be parameterized with other types.

An enum type is a distinct type with named constants. Every enum type has an underlying type, which must be one of the eight integral types. The set of values of an enum type is the same as the set of values of the underlying type.

C# supports single- and multi-dimensional arrays of any type. Unlike the types listed above, array types do not have to be declared before they can be used. Instead, array types are constructed by following a type name with square brackets. For example, int[] is a single-dimensional array of int, int[,] is a two-dimensional array of int, and int[][] is a single-dimensional array of single-dimensional array of int.

Nullable value types also do not have to be declared before they can be used. For each non-nullable value type T there is a corresponding nullable value type T?, which can hold an additional value, null. For instance, int? is a type that can hold any 32-bit integer or the value null.

There are several kinds of *variables* in C#, including fields, array elements, local variables, and parameters. Variables represent storage locations, and every variable has a type that determines what values can be stored in the variable, as shown below.

- Non-nullable value type
  - A value of that exact type
- Nullable value type
  - A null value or a value of that exact type
- object
  - A null reference, a reference to an object of any reference type, or a reference to a boxed value of any value type
- Class type
  - A null reference, a reference to an instance of that class type, or a reference to an instance of a class derived from that class type
- Interface type
  - A null reference, a reference to an instance of a class type that implements that interface type, or a reference to a boxed value of a value type that implements that interface type
- Array type
  - A null reference, a reference to an instance of that array type, or a reference to an instance of a compatible array type
- Delegate type
  - A null reference or a reference to an instance of a compatible delegate type.

Statements

The actions of a program are expressed using *statements*. C# supports several different kinds of statements, a number of which are defined in terms of embedded statements.

A *block* permits multiple statements to be written in contexts where a single statement is allowed. A block consists of a list of statements written between the delimiters { and }.

Declaration statements are used to declare local variables and constants.

Expression statements are used to evaluate expressions. Expressions that can be used as statements include method invocations, object allocations using the new operator, assignments using = and the compound assignment operators, increment and decrement operations using the ++ and -- operators and await expressions.

Selection statements are used to select one of a number of possible statements for execution based on the value of some expression. In this group are the if and switch statements.

Iteration statements are used to execute repeatedly an embedded statement. In this group are the while, do, for, and foreach statements.

Jump statements are used to transfer control. In this group are the break, continue, goto, throw, return, and yield statements.

The try...catch statement is used to catch exceptions that occur during execution of a block, and the try...finally statement is used to specify finalization code that is always executed, whether an exception occurred or not.

The checked and unchecked statements are used to control the overflow-checking context for integral-type arithmetic operations and conversions.

The lock statement is used to obtain the mutual-exclusion lock for a given object, execute a statement, and then release the lock.

The using statement is used to obtain a resource, execute a statement, and then dispose of that resource.

The following lists the kinds of statements that can be used, and provides an example for each.

- Local variable declaration:

```
static void Declarations(string[] args)
{
    int a;
    int b = 2, c = 3;
    a = 1;
    Console.WriteLine(a + b + c);
}
```

Local constant declaration:

```
static void ConstantDeclarations(string[] args)
{
    const float pi = 3.1415927f;
    const int r = 25;
    Console.WriteLine(pi * r * r);
}
```

Expression statement:

```
static void Expressions(string[] args)
{
    int i;
    i = 123;                // Expression statement
    Console.WriteLine(i);   // Expression statement
    i++;                    // Expression statement
    Console.WriteLine(i);   // Expression statement
}
```

If statement:

```
static void IfStatement(string[] args)
{
    if (args.Length == 0)
    {
        Console.WriteLine("No arguments");
    }
    else
    {
        Console.WriteLine("One or more arguments");
    }
}
```

Switch statement:

```
static void SwitchStatement(string[] args)
{
    int n = args.Length;
    switch (n)
    {
        case 0:
            Console.WriteLine("No arguments");
            break;
        case 1:
            Console.WriteLine("One argument");
            break;
        default:
            Console.WriteLine($"{n} arguments");
            break;
    }
}
```

While statement:

```
static void WhileStatement(string[] args)
{
    int i = 0;
    while (i < args.Length)
    {
        Console.WriteLine(args[i]);
        i++;
    }
}
```

Do statement:

```
static void DoStatement(string[] args)
{
    string s;
    do
    {
        s = Console.ReadLine();
        Console.WriteLine(s);
    } while (!string.IsNullOrEmpty(s));
}
```

For statement:

```
static void ForStatement(string[] args)
{
    for (int i = 0; i < args.Length; i++)
    {
        Console.WriteLine(args[i]);
    }
}
```

Foreach statement:

```
static void ForEachStatement(string[] args)
{
    foreach (string s in args)
    {
        Console.WriteLine(s);
    }
}
```

Break statement:

```
static void BreakStatement(string[] args)
{
    while (true)
    {
        string s = Console.ReadLine();
        if (string.IsNullOrEmpty(s))
            break;
        Console.WriteLine(s);
    }
}
```

Continue statement:

```
static void ContinueStatement(string[] args)
{
    for (int i = 0; i < args.Length; i++)
    {
        if (args[i].StartsWith("/"))
            continue;
        Console.WriteLine(args[i]);
    }
}
```

Goto statement:

```
static void GoToStatement(string[] args)
{
    int i = 0;
    goto check;
    loop:
    Console.WriteLine(args[i++]);
    check:
    if (i < args.Length)
        goto loop;
}
```

Return statement:

```
static int Add(int a, int b)
{
    return a + b;
}
static void ReturnStatement(string[] args)
{
   Console.WriteLine(Add(1, 2));
   return;
}
```

Throw and try statements:

```
static double Divide(double x, double y)
{
    if (y == 0)
        throw new DivideByZeroException();
    return x / y;
}
static void TryCatch(string[] args)
{
    try
    {
        if (args.Length != 2)
        {
            throw new InvalidOperationException("Two numbers required");
        }
        double x = double.Parse(args[0]);
        double y = double.Parse(args[1]);
        Console.WriteLine(Divide(x, y));
    }
    catch (InvalidOperationException e)
    {
        Console.WriteLine(e.Message);
    }
    finally
    {
        Console.WriteLine("Good bye!");
    }
}
```

Using statement:

```
static void UsingStatement(string[] args)
{
    using (TextWriter w = File.CreateText("test.txt"))
    {
        w.WriteLine("Line one");
        w.WriteLine("Line two");
        w.WriteLine("Line three");
    }
}
```

Object Oriented Programming Basics

Object-oriented programming (OOP) is a programming language model in which programs are organized around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. Examples of an object can range from physical entities, such as a human being that is described by properties like name and address, down to small computer programs, such as widgets. This opposes the historical approach to programming where emphasis was placed on how the logic was written rather than how to define the data within the logic.

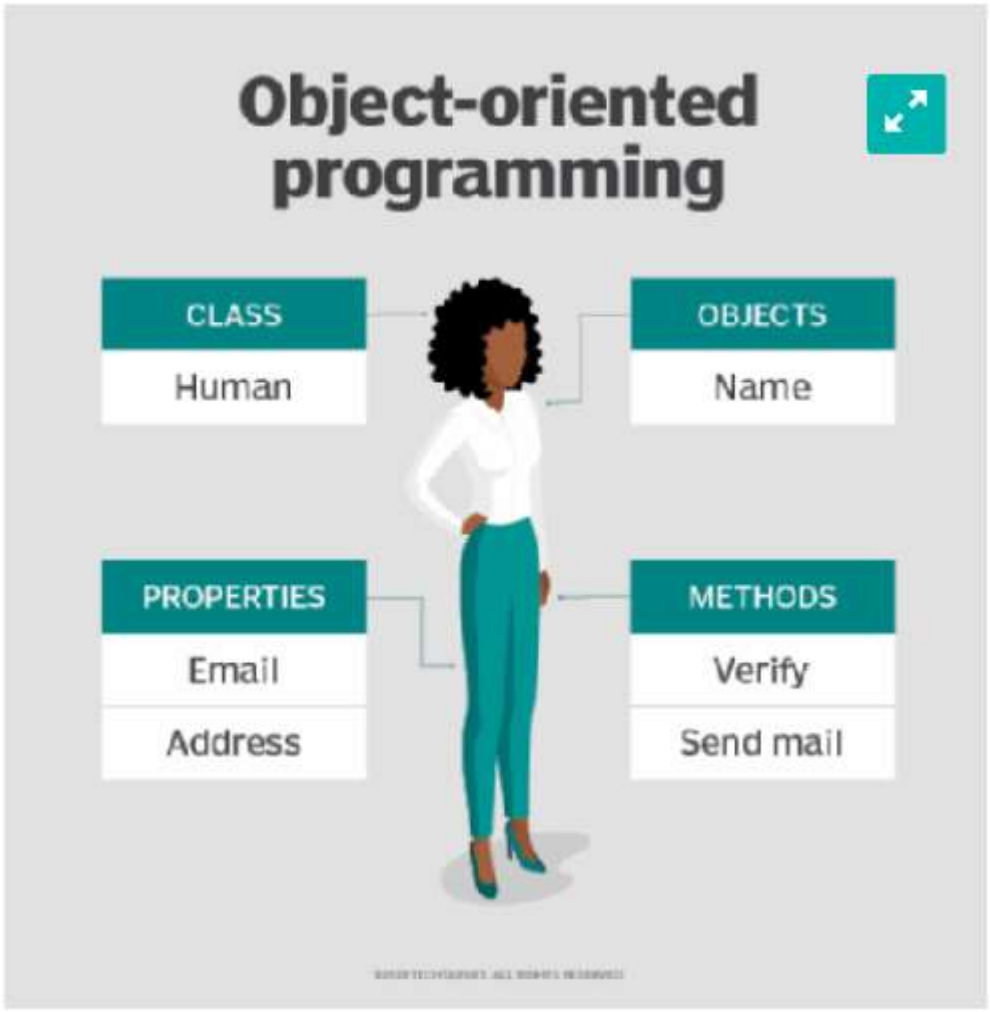
The first step in OOP is to identify all of the objects a programmer wants to manipulate and how they relate to each other, an exercise often known as data modeling. Once an object is known, it is generalized as a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method and objects can communicate with well-defined interfaces called messages.

Simply put, OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained. Due to the organization of an object-oriented program, this method is also conducive to collaborative development where projects can be divided into groups. Additional benefits of OOP include code reusability, scalability and efficiency.

Principles of OOP

Object oriented programming is based on the following principles:

1. Encapsulation- The implementation and state of each object are privately held inside a defined boundary, or class. Other objects do not have access to this class or the authority to make changes but are only able to call a list of public functions, or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.
2. Abstraction- Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. This concept helps developers make changes and additions over time more easily.
3. Inheritance- Relationships and subclasses between objects can be assigned, allowing developers to reuse a common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
4. Polymorphism- Objects are allowed to take on more than one form depending on the context. The program will determine which meaning or usage is necessary for each execution of that object, cutting down on the need to duplicate code.



An example of the conventions in object oriented programming.

The most popular OOP languages are :

- Java
- Python
- C++
- VB.Net

User Controls

The core set of ASP.NET controls is broad and impressive. It includes controls that encapsulate basic HTML tags and controls that provide a rich higher-level model, such as the Calendar, TreeView, and data controls. Of course, even the best set of controls can't meet the needs of every developer. Sooner or later, you'll want to get under the hood, start tinkering, and build your own user interface components.
In .NET, you can plug into the web forms framework with your own controls in two ways. You can develop either of the following:

User controls: A user control is a small section of a page that can include static HTML code and web server controls. The advantage of user controls is that once you create one, you can reuse it in multiple pages in the same web application. You can even add your own properties, events, and methods.

Custom server controls: Custom server controls are compiled classes that programmatically generate their own HTML. Unlike user controls (which are declared like web-form pages in a plaintext file), server controls are always precompiled into DLL assemblies. Depending on how you code the server control, you can render the content from scratch, inherit the appearance and behaviour from an existing web control and extend its features, or build the interface by instantiating and
configuring a group of constituent controls.

In this chapter, you'll explore the first option—user controls. User controls are a great way to standardize repeated content across all the pages in a website. For example, imagine you want to provide a consistent way for users to enter address information on several different pages. To solve this problem, you could create an address user control that combines a group of text boxes and a few related validators. You could then add this address control to any web form and program against it as a single object.

User controls are also a good choice when you need to build and reuse site headers, footers, and navigational aids. (Master pages, which are discussed in Chapter 16, complement user controls by giving you a way to standardize web-page layout.) In all of these examples, you could avoid user controls entirely and just copy and paste the code wherever you need to. However, if you do, you'll run into serious problems once you need to modify, debug, or enhance the controls in the future. Because multiple copies of the user interface code will be scattered throughout your website, you'll have the
unenviable task of tracking down each copy and repeating your changes. Clearly, user controls provide a more elegant, object-oriented approach.

User Control Basics

User control (.ascx) files are similar to ASP.NET web-form (.aspx) files. Like web forms, user controls are composed of a user interface portion with control tags (the .ascx file) and can use inline script or a .cs code-behind file. User controls can contain just about anything a web page can, including static HTML content and ASP.NET controls, and they also receive the same events as the Page object (like Load and PreRender) and expose the same set of intrinsic ASP.NET objects through properties (such as Application, Session, Request, and Response).

The key differences between user controls and web pages are as follows:

- User controls begin with a Control directive instead of a Page directive.
- User controls use the file extension .ascx instead of .aspx, and their code-behind files inherit from the System.Web.UI.UserControl class. In fact, the UserControl class and the Page class both inherit from the same TemplateControl class, which is why they share so many of the same methods and events.
- User controls can't be requested directly by a client browser. (ASP.NET will give a generic "that file type is not served" error message to anyone who tries.) Instead, user controls are embedded inside other web pages.

Creating a Simple User Control

To create a user control in Visual Studio, select Website ➤ Add New Item, and choose the Web User Control template.

The following is the simplest possible user control—one that merely contains static HTML. This user control represents a header bar.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Header.ascx.cs" Inherits="Header" %>
<table width="100%" border="0" style="background-color: Blue">
<tr>
<td style="...">
<b>User Control Test Page</b>
</td>
</tr>
<tr>
<td align="right" style="...">
<b>An Apress Creation © 2008</b>
</td>
</tr>
</table>
```

You'll notice that the Control directive identifies the code-behind class. However, the simple header control doesn't require any custom code to work, so you can leave the class empty:

```
public partial class Header : System.Web.UI.UserControl
{}
```

As with ASP.NET web forms, the user control is a partial class, because it's merged with a separate portion generated by ASP.NET. That automatically generated portion has the member variables for all the controls you add at design time.

Now to test the control, you need to place it on a web form. First, you need to tell the ASP.NET page that you plan to use that user control with the Register directive, which you can place immediately after the Page directive, as shown here:

```
<%@ Register TagPrefix="apress" TagName="Header" Src="Header.ascx" %>
```

This line identifies the source file that contains the user control using the Src attribute. It also defines a tag prefix and tag name that will be used to declare a new control on the page. In the same way that ASP.NET server controls have the <asp: ... > prefix to declare the controls (for example, <asp:TextBox>), you can use your own tag prefixes to help distinguish the controls you've created. This example uses a tag prefix of apress and a tag named Header.

The full tag is shown in this page:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="HeaderTest.aspx.cs"
Inherits="HeaderTest" %>
<%@ Register TagPrefix="apress" TagName="Header" Src="Header.ascx" %>
<html mlns="http://www.w3.org/1999/xhtml">
<head>
      <title>HeaderHost</title>
</head>
<body>
      <form id="Form1" method="post" runat="server">
            <apress:Header id="Header1" runat="server"></apress:Header>
      </form>
</body>
</html>
```

At a bare minimum, when you add a user control to your page, you should give it a unique ID and indicate that it runs on the server, like all ASP.NET controls. Below figure shows the sample page with the custom header.

In Visual Studio, you don't need to code the Register directive by hand. Instead, once you've created your user control, simply select the .ascx file in the Solution Explorer and drag it onto the design area of a web form (not the source view). Visual Studio will automatically add the Register directive for you as well as an instance of the user control tag.
The header control is the simplest possible user control example, but it can already provide some realistic benefits. Think about what might happen if you had to manually copy the header's HTML code into all your ASP.NET pages, and then you had to change the title, add a contact link, or something else.

You would need to change and upload all the pages again. With a separate user control, you just update that one file. Best of all, you can use any combination of HTML, user controls, and server controls on an ASP.NET web form.

Adding Code to a User Control

The previous user control didn't include any code. Instead, it simply provided a useful way to reuse a static block of a web-page user interface. In many cases, you'll want to add some code to your user control creation, either to handle events or to add functionality that the client can access. Just like a web form, you can add this code to the user control class in a <script> block directly in the .ascx file, or you can use a separate .cs code-behind file.

Handling Events

To get a better idea of how this works, the next example creates a simple TimeDisplay user control with some event-handling logic. This user control encapsulates a single LinkButton control. Whenever the link is clicked, the time displayed in the link is updated. The time is also refreshed when the control first loads.

Here's the user control markup:

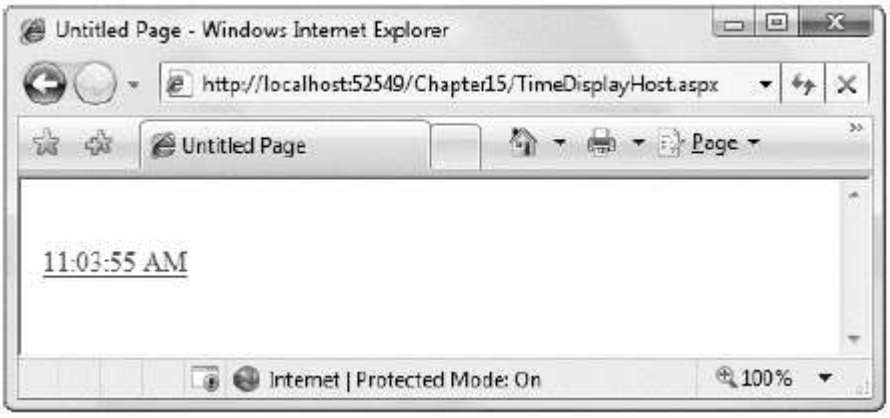
```
<%@ Control Language="c#" AutoEventWireup="true" CodeFile="TimeDisplay.ascx.cs"
Inherits="TimeDisplay" %>
<asp:LinkButton id="lnkTime" runat="server" OnClick="lnkTime_Click" />
```

And here's the corresponding code-behind class:

```
public partial class TimeDisplay : System.Web.UI.UserControl
{
	protected void Page_Load(object sender, EventArgs e)
	{
		if (!Page.IsPostBack)
		RefreshTime();
	}
	protected void lnkTime_Click(object sender, EventArgs e)
	{
		RefreshTime();
	}
	public void RefreshTime()
	{
		lnkTime.Text = DateTime.Now.ToLongTimeString();
	}
}
```

Note that the lnkTime_Click event handler calls a method named RefreshTime(). Because this method is public, the code on the hosting web form can trigger a label refresh programmatically by calling RefreshTime().

Below Figure shows the resulting control.


Untitled Page - Windows Internet Explorer
http://localhost:52549/Chapter15/TimeDisplayHost.aspx
Untitled Page
Page
11:03:55 AM
Internet | Protected Mode: On
100%

Website Navigation

Navigation is a fundamental component of any website. Although it's easy enough to transfer the user from one page to another, creating a unified system of navigation that works across an entire website takes more effort. While you could build your own navigation system with a few links (and a lot of work), ASP.NET has a built-in navigation system that makes it easy.

In this chapter, you'll tackle three core topics:

- **The MultiView and Wizard controls:** These let you boil down a series of steps into a single page. With the help of these controls, you can combine several pages of work into one place, simplifying your navigation needs.

- **The site map model:** This lets you define the navigation structure of your website and bind it directly to rich controls. You'll also learn how to extend this framework to support different types of controls and different site map storage locations.

- **The rich navigational controls:** These include the TreeView and Menu. Although these controls aren't limited to navigation, they're an ideal match. In this chapter, you'll learn about their wide range of features.

Pages with Multiple Views

Most websites split tasks across several pages. For example, if you want to add an item to your shopping cart and take it to the checkout in an e-commerce site, you'll need to jump from one page to another. This is the cleanest approach, and it's easy to program—provided you use some sort of state management technique (from query strings to session state) to transfer information from one page to another.
In other situations, you might want to embed the code for several different pages inside a single page. For example, you might want to provide several views of the same data (such as a grid-based view and a chart-based view) and allow the user to switch from one view to the other without leaving the page. Or, you might want to handle a small multistep task (such as supplying user information for an account sign-up process), without worrying about how to transfer the relevant information between web pages.

In ASP.NET 1.*x*, the only way to model a page with multiple views was to add several Panel controls to a page so that each panel represents a single view or a single step. You could then set the Visible property of each Panel so that you see only one at a time. The problem with this approach is that it clutters your page with extra code for managing the panels. Additionally, it's not very robust—with a minor mistake, you can end up with two panels showing at the same time.
With ASP.NET 4, there's no need to design your own multiple view system from scratch. Instead, you can use one of two higher-level controls that make these designs much easier—the MultiView and the Wizard.

The Multiview Control

The MultiView is the simpler of the two multiple view controls. Essentially, the MultiView gives you a way to declare multiple views and show only one at a time. It has no default user interface—you get only whatever HTML and controls you add.

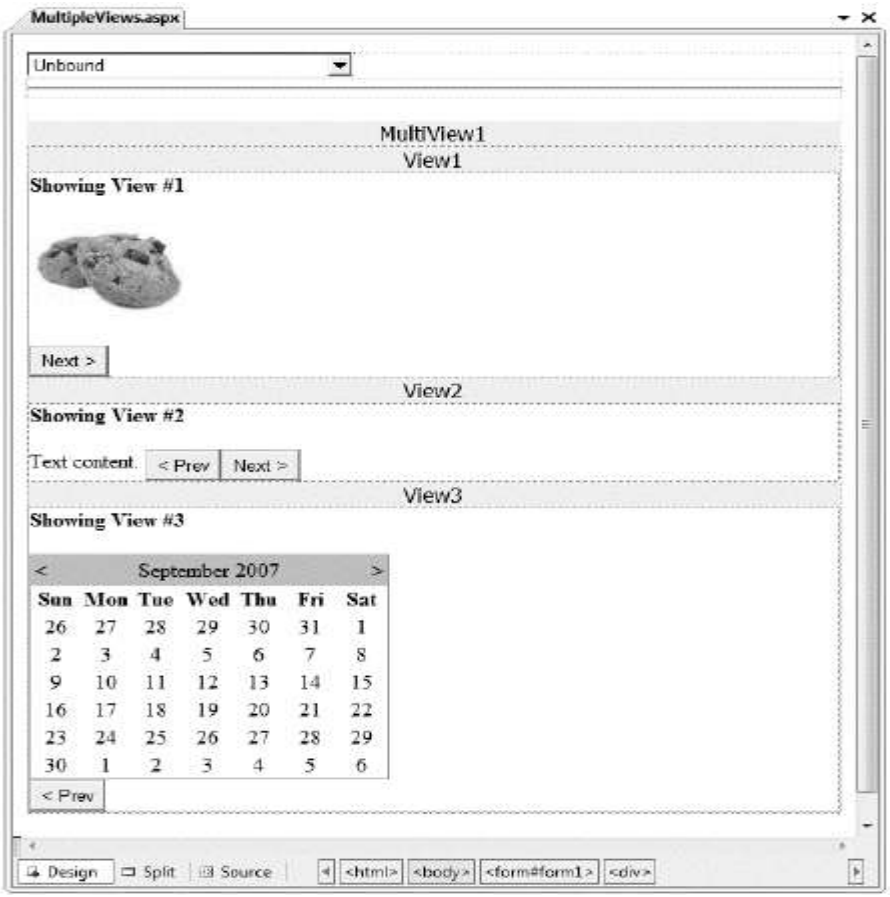
Creating a MultiView is suitably straightforward. You add the <asp:MultiView> tag to your .aspx page file and then add one <asp:View> tag inside it for each separate view.

```
<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>
```

Inside the <asp:View> tag, you add the HTML and web controls for that view.

```
<asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="0">
  <asp:View ID="View1" runat="server">
    <b>Showing View #1<br />
    <br />
    <asp:Image ID="Image1" runat="server"
ImageUrl="./cookies.jpg"/></b>
  </asp:View>
  <asp:View ID="View2" runat="server">
    <b>Showing View #2</b><br />
    <br />
    Text content.
  </asp:View>
  <asp:View ID="View3" runat="server">
    <b>Showing View #3</b><br />
    <br />
    <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
  </asp:View>
</asp:MultiView>
```

Visual Studio shows all your views at design time, one after the other (see Figure 17-1). You can edit these regions in the same way you design any other part of the page.



The MultiView.ActiveViewIndex determines what view will be shown. This is the only view that's rendered in the page. The default ActiveViewIndex value is -1, which means no view is shown. One option is to use a list control that lets users choose from the full list of views. Here's some sample code that binds the list of views to a list box:

```
protected void Page_Load(object sender, EventArgs e)
{
 if (!Page.IsPostBack)
 {
  DropDownList1.DataSource = MultiView1.Views;
  DropDownList1.DataTextField = "ID";
  DropDownList1.DataBind();
 }
}
```

And here's the code that sets the current view based on the list index:

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
  MultiView1.ActiveViewIndex = DropDownList1.SelectedIndex;
}
```

The Performance of MultiView Pages

The most important detail you need to know about 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 you add a control named textBox1 to a view, you can't add another control named textBox1 to another view. In fact, in terms of the page model, there's no real difference between controls you add to a view and controls in the rest of the page. Either way, the controls you create will be accessible through member variables in your 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 you create using the MultiView tend to be heavier than normal pages. That's because the entire control model—including the controls from every view—is created on every postback and persisted to view state. For the most part, this won't be a significant factor, unless you are manipulating a large number of controls programmatically (in which case you might want to turn EnableViewState off for these controls) or you are using several data sources. For example, if you 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, you can leave your controls unbound and binding them programmatically, or canceling the binding process for views that aren't currently visible.

The Wizard Control

The Wizard control is a more glamorous version of the MultiView control. It also supports showing one of several views at a time, but it includes a fair bit of built-in yet customizable behavior, including navigation buttons, a sidebar with step links, styles, and templates.
Usually, wizards represent a single task, and the user moves linearly through them, moving from the current step to the one immediately following it (or the one immediately preceding it in the case of a correction). The ASP.NET Wizard control also supports nonlinear navigation, which means it allows you to decide to ignore a step based on the information the user supplies.
By default, the Wizard control supplies navigation buttons and a sidebar with links for each step on the left. You can hide the sidebar by setting the Wizard.DisplaySideBar property to false. Usually, you'll take this step if you want to enforce strict step-by-step navigation and prevent the user from jumping out of sequence. You supply the content for each step using any HTML or ASP.NET controls.

Wizard Steps

To create a wizard in ASP.NET, you simply define the steps and their content using <asp:WizardStep> tags. Each step takes a few basic pieces of information. The most important ones are listed in the below Table.

Property	Description
Title	The descriptive name of the step. This name is used for the text of the links in the sidebar.
StepType	The type of step, as a value from the WizardStepType enumeration. This value determines the type of navigation buttons that will be shown for this step. Choices include Start (shows a Next button), Step (shows Next and Previous buttons), Finish (shows a Finish and Previous button), Complete (show no buttons and hides the sidebar, if it's enabled), and Auto (the step type is inferred from the position in the collection). The default is Auto, which means that the first step is Start, the last step is Finish, and all other steps are Step.
AllowReturn	Indicates whether the user can return to this step. If false, once the user has passed this step, the user will not be able to return. The sidebar link for this step will have no effect, and the Previous button of the following step will either skip this step or be hidden completely (depending on the AllowReturn value of the preceding steps).

The following wizard contains four steps that, taken together, represent a simple survey. The StepType adds a Complete step at the end, with a summary. The navigation buttons and sidebar links are added automatically.

```
<asp:Wizard ID="Wizard1" runat="server" Width="467px" BackColor="#EFF3FB"
BorderColor="#B5C7DE" BorderWidth="1px">
      <WizardSteps>
            <asp:WizardStep ID="WizardStep1" runat="server" Title="Personal">
                  <h3>Personal Profile</h3>
                  Preferred Programming Language:
                  <asp:DropDownList ID="lstLanguage" runat="server">
                  <asp:ListItem>C#</asp:ListItem>
                  <asp:ListItem>VB</asp:ListItem>
                  <asp:ListItem>J#</asp:ListItem>
                  <asp:ListItem>Java</asp:ListItem>
                  <asp:ListItem>C++</asp:ListItem>
                  <asp:ListItem>C</asp:ListItem>
                  </asp:DropDownList>
                  <br />
            </asp:WizardStep>
            <asp:WizardStep ID="WizardStep2" runat="server" Title="Company">
                  <h3>Company Profile</h3>
                  Number of Employees: <asp:TextBox ID="txtEmpCount"
                  runat="server"/>
                  Number of Locations: <asp:TextBox ID="txtLocCount"
                  runat="server"/>
            </asp:WizardStep>
            <asp:WizardStep ID="WizardStep3" runat="server" Title="Software">
                  <h3>Software Profile</h3>
                  Licenses Required:
                  <asp:CheckBoxList ID="lstTools" runat="server">
                  <asp:ListItem>Visual Studio 2008</asp:ListItem>
                  <asp:ListItem>Office 2007</asp:ListItem>
                  <asp:ListItem>Windows Server 2008</asp:ListItem>
                  <asp:ListItem>SQL Server 2008</asp:ListItem>
                  </asp:CheckBoxList>
            </asp:WizardStep>
            <asp:WizardStep ID="Complete" runat="server" Title="Complete"
            StepType="Complete">
                  <br />
                  Thank you for completing this survey.<br />
                  Your products will be delivered shortly.<br />
            </asp:WizardStep>
      </WizardSteps>
</asp:Wizard>
```

Unlike the MultiView control, you can see only one step at a time on the design surface of your web page in Visual Studio. To choose which step you're currently designing, select it from the smart tag as shown in the below figure. But be warned—every time you do, Visual Studio changes the Wizard.ActiveStepIndex property to the step you choose. Make sure you set this back to 0 before you run your application so it starts at the first step.

Wizard Events

You can write the code that underpins your wizard by responding to several events (as listed in the Table below).

Event	Description
ActiveStepChanged	Occurs when the control switches to a new step (either because the user has clicked a navigation button or your code has changed the ActiveStepIndex property).
CancelButtonClick	Occurs when the Cancel button is clicked. The cancel button is not shown by default, but you can add it to every step by setting the Wizard.DisplayCancelButton property. Usually, a cancel button exits the wizard. If you don't have any cleanup code to perform, just set the CancelDestinationPageUrl property, and the wizard will take care of the redirection automatically.
FinishButtonClick	Occurs when the Finish button is clicked.
NextButtonClick and PreviousButtonClick	Occurs when the Next or Previous button is clicked on any step. However, because there is more than one way to move from one step to the next, it's better to handle the ActiveStepChanged event.
SideBarButtonClick	Occurs when a button in the sidebar area is clicked.

On the whole, two wizard programming models exist:

Commit-as-you-go: This makes sense if each wizard step wraps an atomic operation that can't be reversed. For example, if you're processing an order that involves a credit card authorization followed by a final purchase, you can't allow the user to step back and edit the credit card number. To support this model, you set the AllowReturn property to false on some or all steps, and you respond to the ActiveStepChanged event to commit changes for each step.

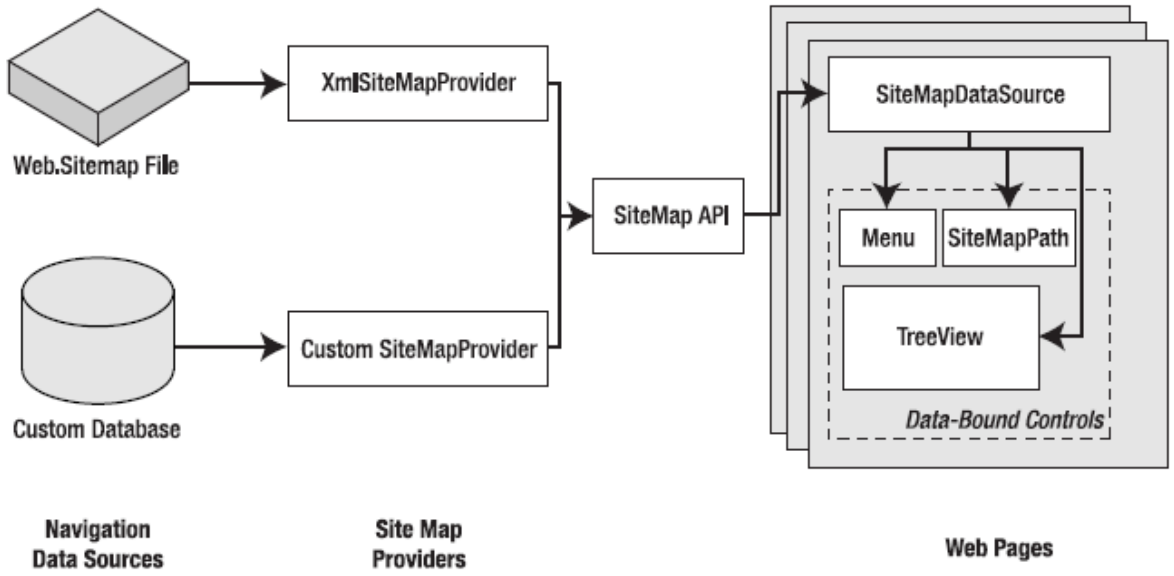
Commit-at-the-end: This makes sense if each wizard step is collecting information for an operation that's performed only at the end. For example, if you're collecting user information and plan to generate a new account once you have all the information, you'll probably allow a user to make changes midway through the process. You execute your code for generating the new account when the wizard is finished by reacting to the FinishButtonClick event.

Site Maps

If your website has more than a handful of pages, you'll probably need some sort of navigation system to let the user move from one page to the next. You can also use master pages to define a template for your site that includes a navigation bar. However, it's still up to you to fill this navigation bar with content.
Obviously, you can use the ASP.NET toolkit of controls to implement almost any navigation system, but it still requires you to perform all the hard work. Fortunately, ASP.NET includes a set of navigation features that you can use to dramatically simplify the task.
As with all the best ASP.NET features, ASP.NET navigation is flexible, configurable, and pluggable. It consists of three components:

- A way to define the navigational structure of your website. This part is the XML site map, which is (by default) stored in a file.
- A convenient way to parse the site map file and convert its information into a suitable object model. This part is performed by the SiteMapDataSource control and the XmlSiteMapProvider.
- 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 is provided through the controls you bind to the SiteMapDataSource control, which can include breadcrumb links, lists, menus, and trees.

You can customize or extend each of these ingredients separately. For example, if you want to change the appearance of your navigation controls, you simply need to bind different controls to the SiteMapDataSource. On the other hand, if you want to read a different format of site map information or read it from a different location, you need to change your site map provider.



Defining a Site Map

The starting point in site map-based navigation is the site map provider. ASP.NET ships with a single site map provider, named XmlSiteMapProvider, which is able to retrieve site map information from an XML file. If you want to retrieve a site map from another location or in a custom format, you'll need to create your own site map provider—a topic covered in the section "Creating a Custom SiteMapProvider."

The XmlSiteMapProvider looks for a file named Web.sitemap in the root of the virtual directory. Like all site map providers, its task is to extract the site map data and create the corresponding SiteMap object. This SiteMap object is then made available to other controls through the SiteMapDataSource.

To try this, you need to begin by creating a Web.sitemap file and defining the website structure using the <siteMap> and <siteMapNode> elements. To add a site map using Visual Studio, choose Website ➤ Add New Item (or Project ➤ Add New Item in a web project), choose the Site Map template, and then click Add.

Here's the bare-bones structure that the site map file uses:

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

To be valid, your site map must begin with the root <siteMap> node, followed by a single <siteMapNode> element, representing the default home page. You can nest other <siteMapNode> elements in the root <siteMapNode> as many layers deep as you want. Each site map node should have a title, description, and URL, as shown here:

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

In this example, the URL uses the ./ relative path syntax, which indicates the root of the web application. This style isn't necessary, but it is strongly recommended, as it ensures that your site map links are interpreted correctly regardless of the current folder.

You can now use the <siteMapNode> to create a site map. The only other restriction is that you can't create two site map nodes with the same URL.

Here's a sample site map:

```
<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 title="Software" description="Software choices" url="./Software.aspx"
/>
</siteMapNode>
<siteMapNode title="Services" description="Services we offer"
url="./Services.aspx">
<siteMapNode title="Training" description="Training classes" url="./Training.aspx"
/>
<siteMapNode title="Consulting" description="Consulting services"
url="./Consulting.aspx" />
<siteMapNode title="Support" description="Support plans" url="./Support.aspx" />
</siteMapNode>
</siteMapNode>
</siteMap>
```

Binding to a Site Map

Once you've defined the Web.sitemap file, you're ready to use it in a page. This is a great place to use master pages so that you can define the navigation controls as part of a template and reuse them with every page. Here's how you might define a basic structure in your master page that puts navigation controls on the left and creates the SiteMapDataSource that provides navigational information to other controls:

```
<form id="form1" runat="server">
       <table>
              <tr>
              <td style="width: 226px;vertical-align: top;">
                 <!-- Navigation controls go here. -->
              </td>
              <td style="vertical-align: top;">
                 <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server" />
              </td>
              </tr>
       </table>
       <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
</form>
```

Then you can create a child page with some simple static content:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
runat="Server">
       <br />

       <br />
       Default.aspx page (home).
</asp:Content>
```

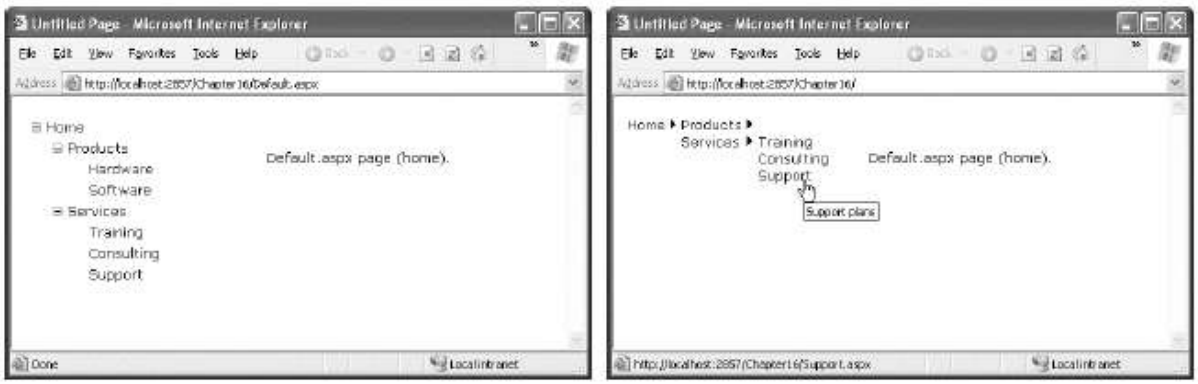
The only remaining task is to choose the controls you want to use to display the site map data. One all-purpose solution is the TreeView control. You can add the TreeView and bind it to the SiteMapDataSource in the master page using the DataSourceID, as shown here:

```
<asp:TreeView ID="treeNav" runat="server" DataSourceID="SiteMapDataSource1" />
```

Alternatively, you could use the fly-out Menu control just as easily:

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1" />
```

Below figure shows both the options:



Creating a Custom SiteMapProvider

To really change how the ASP.NET navigation model works, you need to create your own site map provider. You might choose to create a custom site map provider for several reasons:

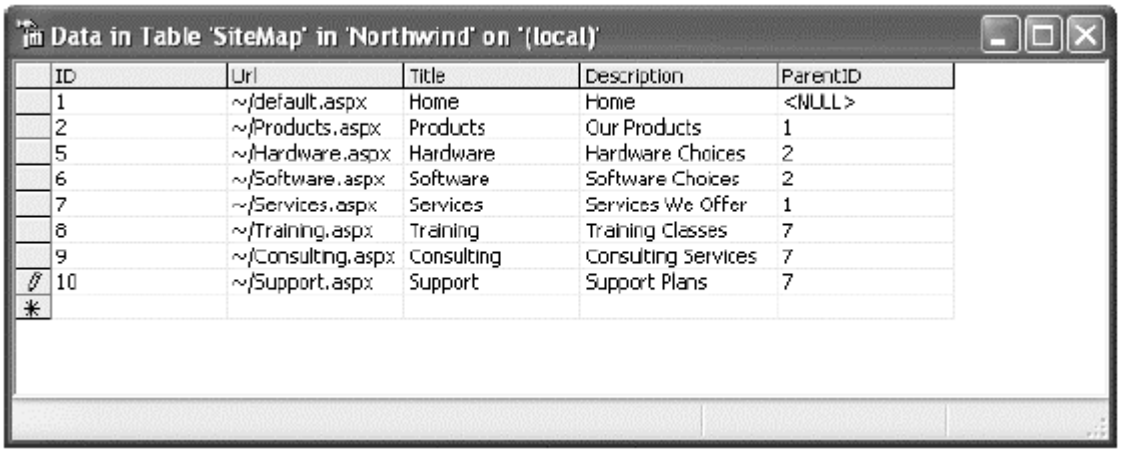
- You need to store site map information in a different data source (such as a relational database).
- You need to store site map information with a different schema from the XML format expected by ASP.NET. This is most likely if you have an existing system in place for storing site maps.
- You need a highly dynamic site map that's generated on the fly. For example, you might want to generate a different site map based on the current user, the query string parameters, and so on.
- You need to change one of the limitations in the XmlSiteMapProvider implementation. For example, maybe you want the ability to have nodes with duplicate URLs.

You have two choices when implementing a custom site map provider. All site map providers derive from the abstract base class SiteMapProvider in the System.Web namespace. You can derive from this class to implement a new provider from scratch. However, if you want to keep the same logic but use a different data store, just derive from the StaticSiteMapProvider class instead. It gives you a basic implementation of many methods, including the logic for node storing and searching.

In the following sections, you'll see a custom provider that lets you store site map information in a database.

Storing Site Map Information in a Database

In this example, all navigation links are stored in a single database table. Because databases don't lend themselves easily to hierarchical data, you need to be a little crafty. In this example, each navigation link is linked to a parent link in the same table, except for the root node. This means that although the navigational links are flattened into one table, you can re-create the right structure by starting with the home page and then searching for the subset of rows at each level.
Figure below shows the SiteMap table with some sample data that roughly duplicates the site map you saw earlier in this chapter.


Data in Table 'SiteMap' in 'Northwind' on '(local)'

ID	Url	Title	Description	ParentID
1	~/default.aspx	Home	Home	<NULL>
2	~/Products.aspx	Products	Our Products	1
5	~/Hardware.aspx	Hardware	Hardware Choices	2
6	~/Software.aspx	Software	Software Choices	2
7	~/Services.aspx	Services	Services We Offer	1
8	~/Training.aspx	Training	Training Classes	7
9	~/Consulting.aspx	Consulting	Consulting Services	7
10	~/Support.aspx	Support	Support Plans	7

In this solution, the site map provider won't access the table directly. Instead, it will use a stored procedure. This gives some added flexibility and potentially allows you to store your navigation information with a different schema, as long as you return a table with the expected column names from your stored procedure.

Here's the stored procedure used in this example:

```
CREATE PROCEDURE GetSiteMap AS
SELECT * FROM SiteMap ORDER BY ParentID, Title
```

Adding Sorting

Currently, the SqlSiteMapProvider returns the results ordered alphabetically by title. This means the About page always appears before the Contact Us page. This make sense for a quick test, but it isn't practical in a real site, where you probably want the ability to control the order in which pages appear.
Fortunately, an easy solution exists. In fact, you don't even need to touch the SqlSiteMapProvider code. All you need to do is introduce a new field in the SiteMap table (say, OrdinalPosition) and modify the GetSiteMap procedure to use it:

```
ALTER PROCEDURE GetSiteMap AS
SELECT * FROM SiteMap ORDER BY ParentID, OrdinalPosition, Title
```

First, records are sorted into groups based on the parent (which node they fall under). Next, they're ordered according to the OrdinalPosition values, if you've supplied them. Finally, they're sorted by title.

DATA BASE : ADO.NET

Contents

11.0 INTRODUCTION TO ADO.NET

Microsoft introduced ADO (ActiveX Data Objects), a data access technology for connecting to databases in 1996.ADO.NET is an extension of ADO, and it enables you to connect to and work with databases from within the managed environment of .NET.ADO is a language-neutral object model that is the keystone of Microsoft's Universal Data Access strategy. ADO.NET is an integral part of the .NET Compact Framework, providing access and modify to relational data, XML documents, and application data.

ADO.NET supports a variety of development needs. You can create database-client applications and middle-tier business objects used by applications, tools, languages or Internet browsers. ADO.NET provides data access services in the Microsoft .NET platform. ADO.NET is an object oriented data access technology that supports both connected and disconnected modes of operation.

You can use ADO.NET to perform CRUD(Create, Read, Update and Delete).You can use ADO.NET to access data by using the new .NET Framework data providers which are:

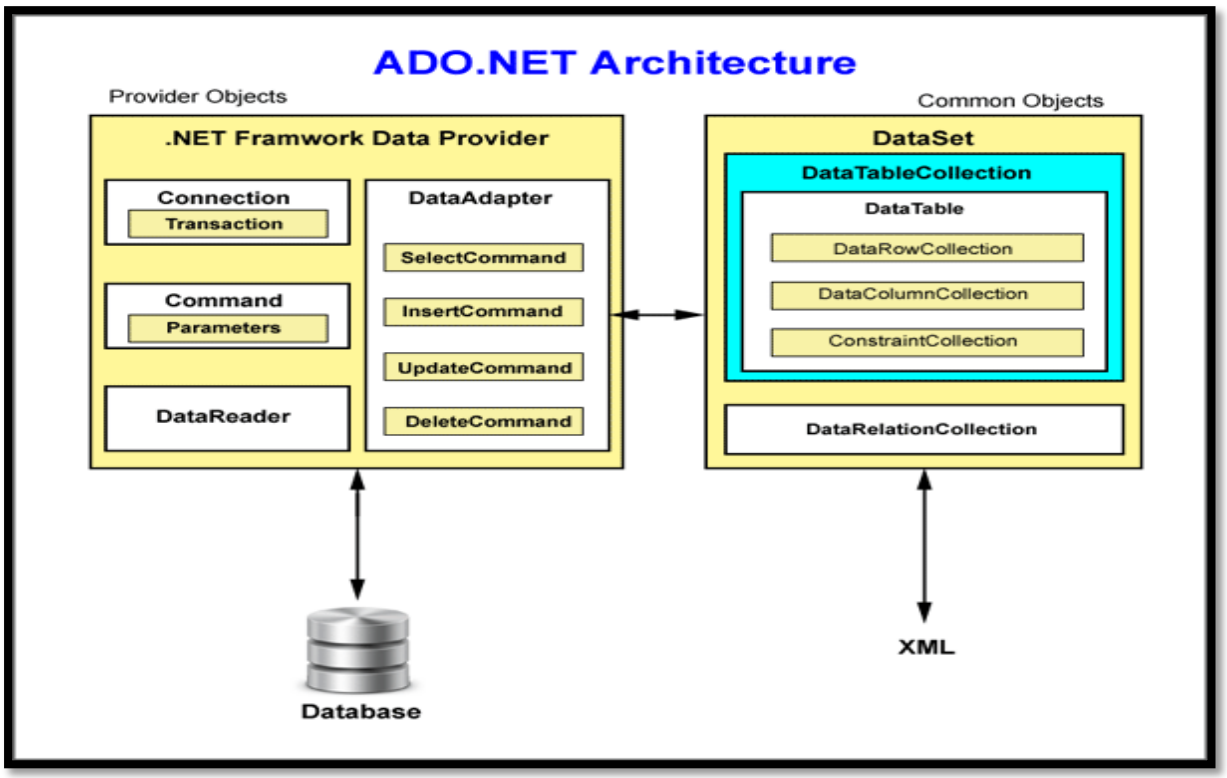
1. Data Provider for SQL Server (System.Data.SqlClient).
2. Data Provider for OLEDB (System.Data.OleDb).
3. Data Provider for ODBC (System.Data.Odbc).
4. Data Provider for Oracle (System.Data.OracleClient).

ADO.NET is a set of classes that expose data access services to the .NET developer. The ADO.NET classes are found inSystem.Data.dll and are integrated with the XML classes in System.Xml.dll.

11.1 ADO.NET ARCHITECTURE

ADO.NET consists of a set of objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational database through a common set of components .

System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.



11.1.1 Components of ADO.NET Architecture

DataSet

DataSet can work in both connected and disconnected modes of ADO.NET. DataSet is the core component of the disconnected architecture of ADO.NET. It is an in-memory representation of a database, provide consistent relational programming model irrespective of the source of the data, which has been read into it. DataSet contains one or more tables. DataAdapter is used to get data in DataSet.

DataTable

In ADO.NET, DataTable objects are used to represent the tables in a DataSet. A DataTable is an in-memory representation of a single database table which has collection of rows and columns. DataTable fetches only one TableRow at a time DataSet. You and add or delete required columns to the DataTable by using the commands.

DataView

A DataView provides a customized view of DataTable. You can use it to sort or filter a rows of a DataSet.

Data Provider

A data provider provides access to the database. The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider, OLEDB Data Provider and ODBC Data provider. A data provider encapsulates the protocols that are needed to make connection, and perform CRUD operations with database.

Connection Object

The connection object is used to establish a connection to the database. It carries required authentic information like username and password in the connection string and opens a connection.
You can use the following connection object

- OLE DB – OleDbConnection
- SQL Server – SqlConnection
- ODBC – OdbcConnection
- Oracle – OracleConnection

Command Object

The command object is used to send SQL statements to the database in order to execute CRUD operation. The SQL queries can be in the Form of Inline text, Stored Procedures or direct Table access. Commands are used to insert data, retrieve data, and execute store procedures and other database objects. Depending on the underlying database in use, you can use the OracleCommand, SqlCommand, OleDbCommand and OdbcCommand objects in ADO.NET.

11.2 PROPERTIES OF SQLCOMMAND CLASS

The properties associated with SqlCommand class are shown in the Table 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 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: ▪ Text (SQL Statement) ▪ StoredProcedure (Stored Procedure Name) ▪ 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.

Execute Methods that can be called from a Command Object.

Property	Description
ExecuteNonQuery	This method executes the command specifies and returns the number of rows affected.
ExecuteReader	The ExecuteReader method executes the command specified and returns an instance of instance of SqlDataReader class.
ExecuteScalar	This method executes the command specified and returns the first column of first row of the result set. The remaining rows and column 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

11.2.1 DataReader

A DataReader is connected, forward-only, read-only stream of data that is used to read a sequential collection of records from a database. The DataReader cannot be created directly from code, they can created only by calling the ExecuteReader method of a Command Object. It is much faster than a DataSet but requires an open connection.

11.2.2 DataAdapter

The DataAdapter is used in the disconnected mode of ADO.NET. DataAdapter acts as a bridge between DataSet and database. DataAdapter object is used to read the data from the

database and bind that data to dataset. DataAdapter resolves the changes made to the DataSet back to the database.

DataAdapter provide two methods : Fill and Update

The Fill method populates a DataSet instance with data from the database. The Update method is used to update the database with data contained in a DataSet. The DataAdapter provides the SelectCommand, InsertCommand, UpdateCommand and DeleteCommand command objects to perform CURD operations.

11.3 CONNECTION STRING

A connection string provides the information that a provider needs to communicate with a particular database. The Connection String includes parameters such as the name of the driver, Server name and Database name , as well as security information such as user name and password.

An ADO.NET Data Provider is a class that can communicate with a specific type of database or data store. Usually Data Providers use a connection string containing a collection of parameters to establish the connection with the database through applications.

The .NET Framework provides mainly three data providers, they are

1. .NET DataProvider(SQL Server);
2. OLEDB
3. ODBC

Syntax

connectionString="Data Source=(LocalDB)\v11.0;AttachDbFileName=|DataDirectory|\DatabaseFileName.mdf;Initial Catalog=DatabaseName;Integrated Security=True;MultipleActiveResultSets=True" />

In a web application we can specify a database connection string in one of the following two ways.

- Specify it in the web.config file.
- Create a common class file for the connection string.

Connection string parameters

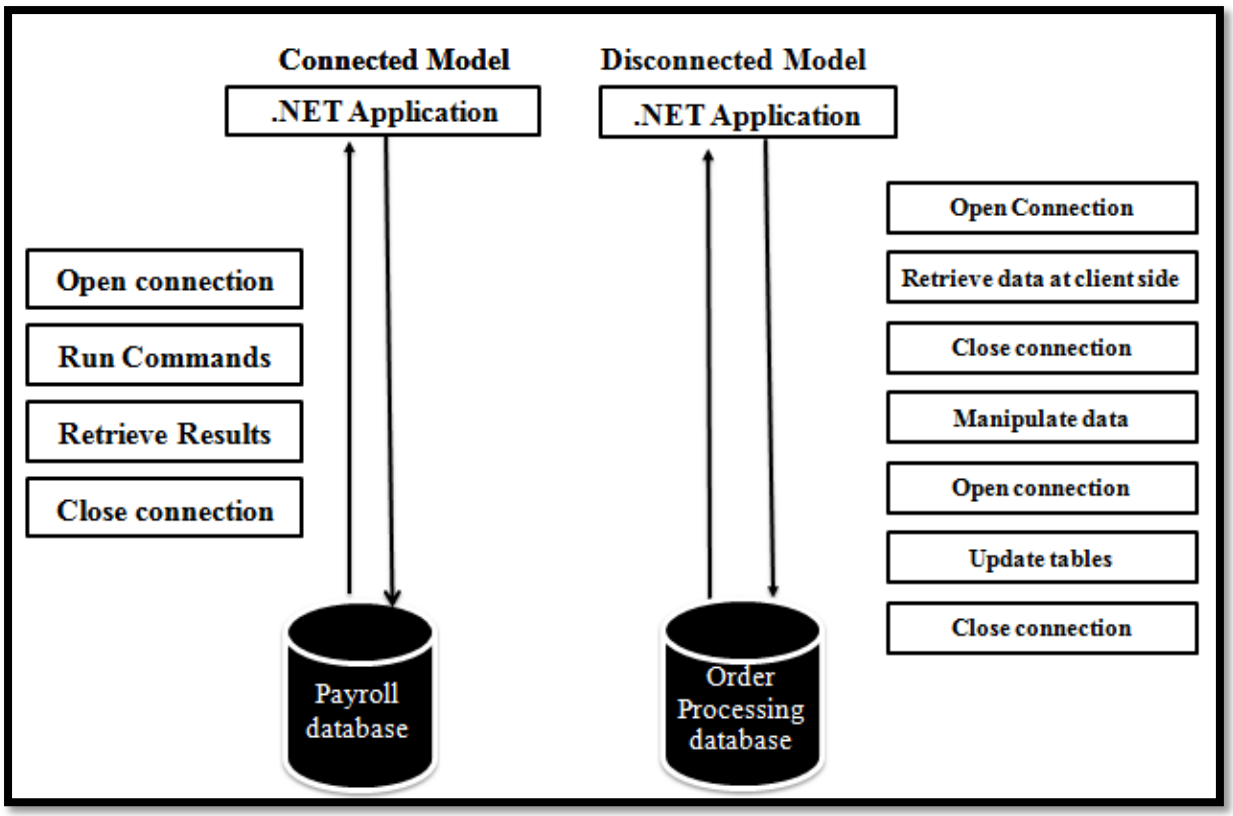
Sr No	Parameter	Description
1	AttachDBFilename /Extended Properties /Initial File Name	The name of the primary database file, including the full path name of an attachable database. AttachDBFilename is only supported for primary data files with an .mdf extension. If the value of the AttachDBFileName key is specified in the connection string, the database is attached and becomes the default database for the connection.

2	Connect Timeout / Connection Timeout	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error. Default time out is 15 seconds.
3	Data Source/ Server/ Address	The name or network address of the instance of SQL Server to which to connect. The port number can be specified after the server name: server=tcp:servername, portnumber
4	Initial Catalog / Database	The name of the database. The database name can be 128 characters or less.
5	Integrated Security	When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication.
6	Persist Security Info	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.
7	User ID	The SQL Server login account. Not recommended. To maintain a high level of security, we strongly recommend that you use the Integrated Security or Trusted_Connection keywords instead. SqlCredential is a more secure way to specify credentials for a connection that uses SQL Server Authentication. The user ID must be 128 characters or less.
8	Password	The password for the SQL Server account logging on. Not recommended. To maintain a high level of security, we strongly recommend that you use the Integrated Security or Trusted_Connection keyword instead.
9	User Instance	A value that indicates whether to redirect the connection from the default SQL Server Express instance to a runtime-initiated instance running under the account of the caller.
10	Application Name	The name of the application, or '.NET SQLClient Data Provider' if no application name is provided.

11.4 CONNECTED AND DISCONNECTED MODE OF ADO.NET ARCHITECTURE

ADO.NET provides mainly the following two types of architectures-

1. Connected Architecture
2. Disconnected Architecture



11.4.1Connected Architecture

In the connected architecture, connection with a data source is kept open constantly for data access as well as data manipulation operations.

The ADO.NET Connected architecture considers mainly three types of objects.

- SqlConnection con;
- SqlCommand cmd;
- SqlDataReader dr;

11.4.1.1 Direct Data Access

The most straightforward way to interact with a database is to use direct data access. When you use direct data access, you're in charge of building a SQL command and executing it. You use commands to query, insert, update, and delete information.

When you query data with direct data access, you don't keep a copy of the information in memory. Instead, you work with it for a brief period of time while the database connection is open, and then close the connection as soon as possible. This is different than disconnected data access, where you keep a copy of the data in the DataSet object so you can work with it after the database connection has been closed. The direct data model is well suited to ASP.NET web pages, which don't need to keep a copy of their data in memory for long periods of time.

When you work with ADO.NET in connected mode, you follow these steps:

1. Create a connection
2. Open a connection
3. Create a command object
4. Execute SQL statements
5. Close the connection

11.4.1.2 Creating the connection:

To create a connection to the database, you need to use the connection class appropriate for the underlying database.

You also need a connection string that contains the database credentials of the database you are connecting to.

Example

You need to import these two namespace before using any ado.net connection.

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

SqlConnection con = new SqlConnection("Data Source=hp; Initial Catalog = zaidi; Integrated Security=True");
```

Opening the connection:

To open the connection in preceding section, you need to use the Open method on the connection object.

Example

```
con.Open();
```

Creating the command:

To create the command object, use the class that corresponds to the database you are using. Here we are using SQL Server as the database, we should use the SqlCommand class

Example

```
SqlCommand  cmd = new SqlCommand();
```

Execute SQL statements

To execute queries using the command object, call the appropriate command object method.

Example

```
cmd.CommandText = "select name,class,mobile from student";
```

Closing the connection

Once we are done with all the operations, we should close the connection to the database using the close method on the connection object.

Example

```
con.Close();
```

11.4.1.3 Inserting, Updating, Deleting and Searching in connected mode

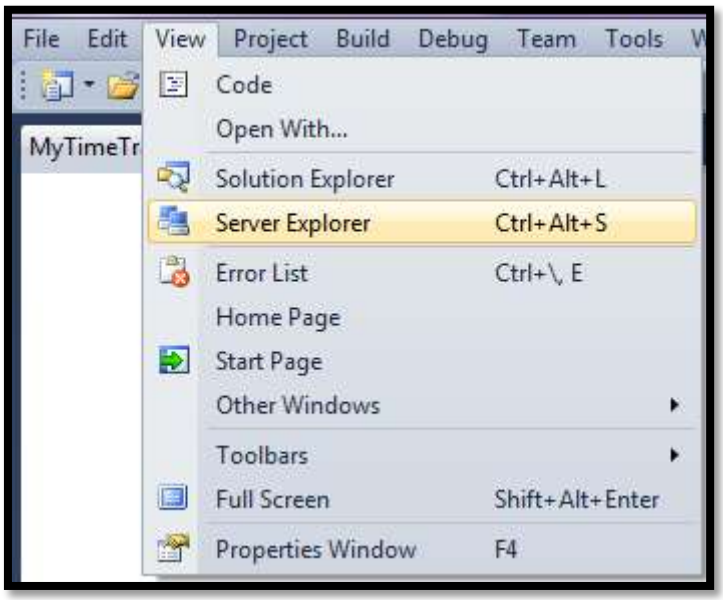
In the connected mode of operation, the connection to the database remains open.

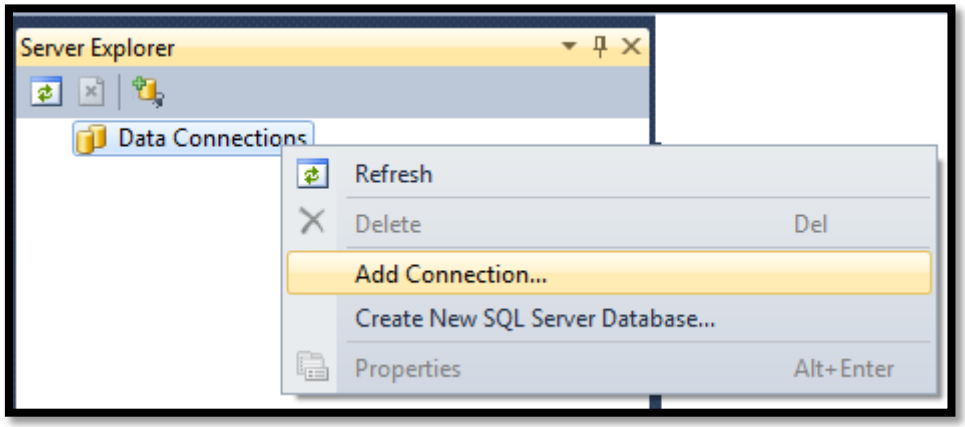
To perform insert, update or delete operations, you need to use the ExecuteQuery method on the command object.

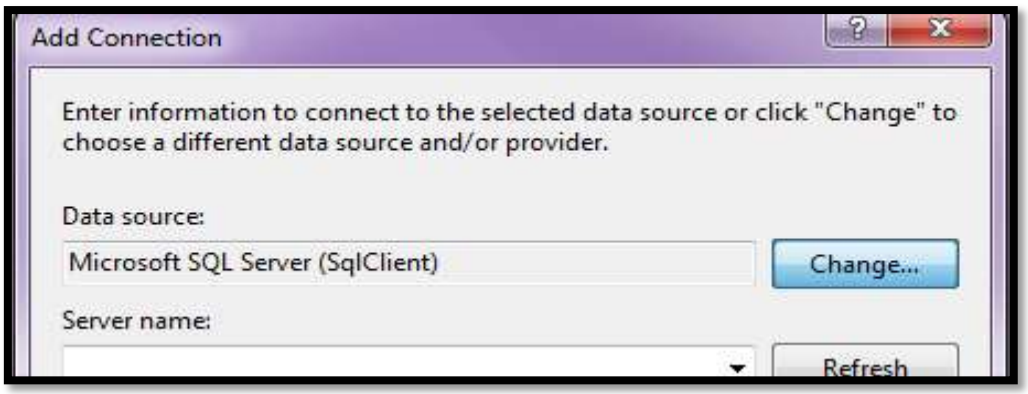
We are performing simple operations like insert, update , delete and search operations in a Web Forms application.

The following are the basic steps of performing insert, update , delete and search operations on database:-

1. At first we should have a Database. So create a database in Microsoft SQL Server.
   In this example our database name is "zaidi" and database table is "student" which has five columns as "no", "name", "class", “mobile” and "course", no column is a primary key of the table.
2. Create a connection in visual studio 2010
   The first thing we want to do is add a data connection in the Visual Studio 2010 Servers Explorer Windows.
   Server Explorer allows us to see the contents of the database. It provides tree structure to explore database files.
   Right click on "Data Connections" and click on "Add New Connection".

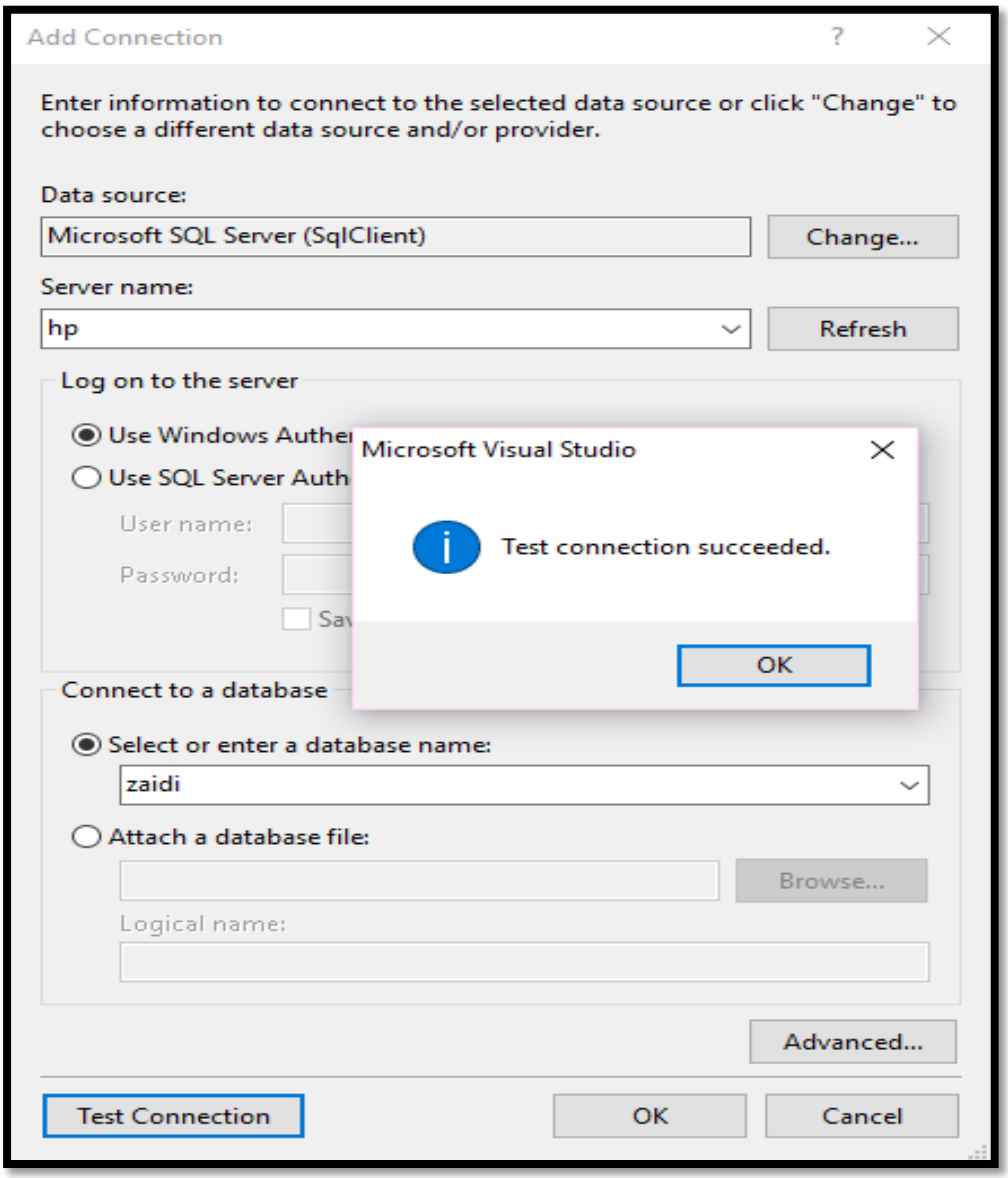




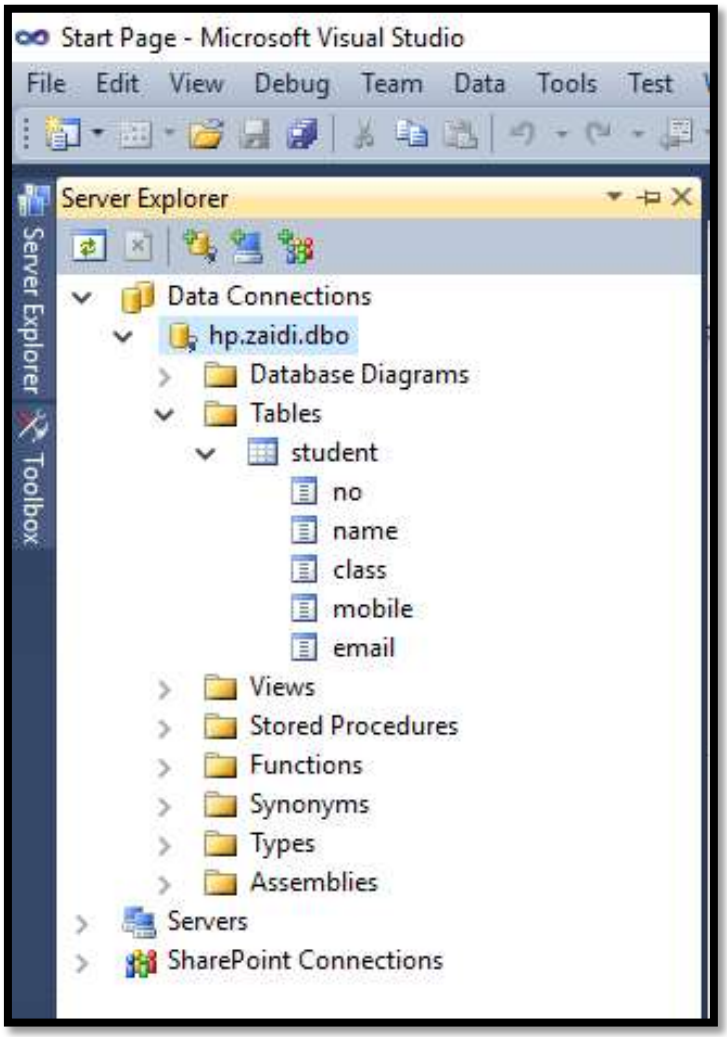


Enter the required connection info in the "Add Connection" dialog and click "Test Connection" to test the connection.

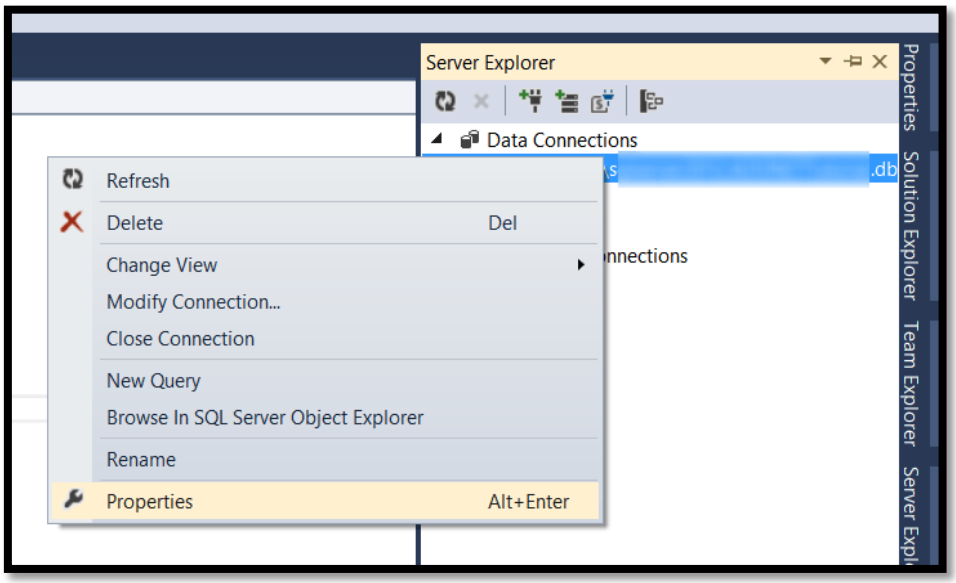
If that succeeds, click OK.

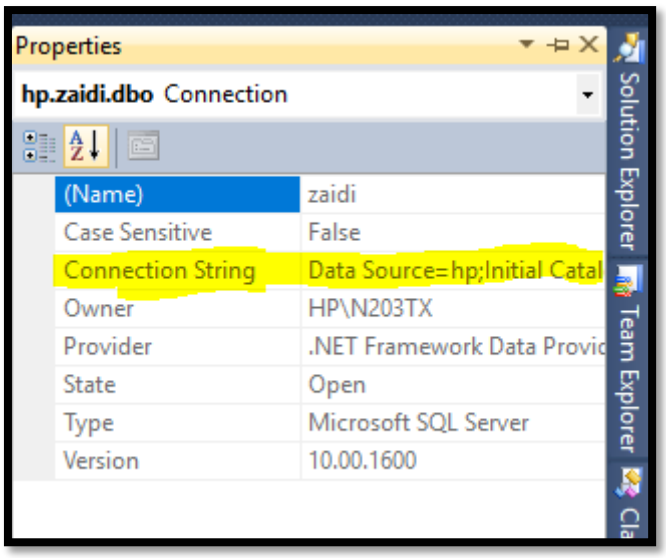


After you add the data connection you will be able to see the connection in the Data Connections tree:



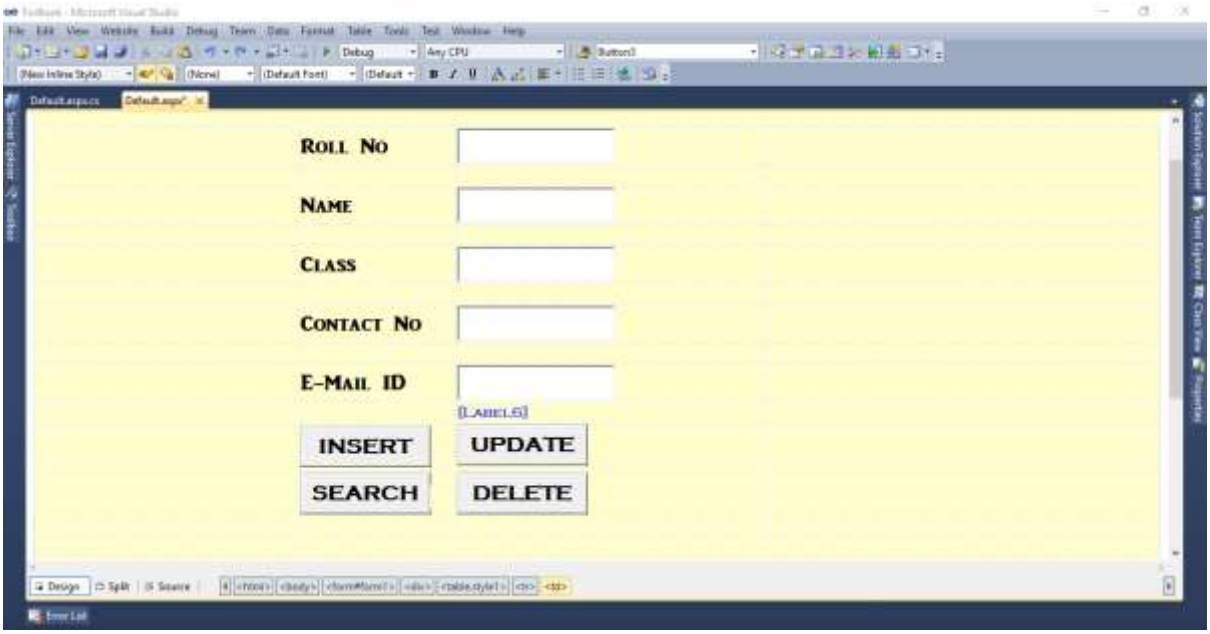
1. Create a command.
2. Specify connection string to the connection.
   Right click on databse.





3. Specify connection that the command will use.
4. Specify the insert/update/delete and other statements for the CommandText of the command.
5. Add value to the command parameters (if any)
6. Open connection
7. Execute the commands
8. Close connection

Create a web forms application and design the following web page



Double click on the button to generate the even handler for the buttons and use the following code for insert, update, delete and search operations.

Default.aspx code

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title> Insert Update Delete By Haider Zaidi </title>
</head>
<body>
   <form id="form1" runat="server">
   <div>

      <table class="style1">
            <td class="style3">
               <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Names="AR
                JULIAN" Font-Size="X-Large" Text="Roll No"></asp:Label>
            </td>
            <td class="style4">
               <asp:TextBox ID="TextBox1" runat="server" Font-Bold="True"
                  Font-Names="AR JULIAN" Font-Size="Medium" Height="40px"
                  Width="179px"></asp:TextBox>
            </td>
            <td class="style3">
               <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Names="AR
              JULIAN" Font-Size="X-Large" Text="Name"></asp:Label>
            </td>
            <td class="style4">
               <asp:TextBox ID="TextBox2" runat="server" Font-Bold="True"
                  Font-Names="AR JULIAN" Font-Size="Medium" Height="40px"
                  Width="179px"></asp:TextBox>
            </td>
            <td class="style3">
               <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Names="AR
               JULIAN" Font-Size="X-Large" Text="Class"></asp:Label>
            </td>
            <td class="style4">
               <asp:TextBox ID="TextBox3" runat="server" Font-Bold="True"
                  Font-Names="AR JULIAN" Font-Size="Medium" Height="40px"
                  Width="179px"></asp:TextBox>
            </td>
            <td class="style3">
               <asp:Label ID="Label4" runat="server" Font-Bold="True" Font-Names="AR
               JULIAN" Font-Size="X-Large" Text="Contact No"></asp:Label>
            </td>
            <td class="style4">
               <asp:TextBox ID="TextBox4" runat="server" Font-Bold="True"
                  Font-Names="AR JULIAN" Font-Size="Medium" Height="40px"
                  Width="179px"></asp:TextBox>
            </td>
            <td class="style3">
```

```
            <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-Names="AR
          JULIAN" Font-Size="X-Large" Text="E-Mail ID"></asp:Label>
          </td>
          <td class="style4">
            <asp:TextBox ID="TextBox5" runat="server" Font-Bold="True"
              Font-Names="AR JULIAN" Font-Size="Medium" Height="40px"
              Width="179px"></asp:TextBox>
          </td>
          <td class="style4">
            <asp:Label ID="Label6" runat="server" Font-Bold="True"
              Font-Names="Copperplate Gothic Light" Font-Size="Medium"
              ForeColor="#3333FF"></asp:Label>
          </td>
          <td class="style3">
            <asp:Button ID="InsertButton" runat="server" Font-Bold="True"
              Font-Names="Copperplate Gothic Bold" Font-Size="X-Large"
Height="47px" onclick="InsertButton_Click" Text="INSERT" Width="148px" />
          </td>
          <td class="style4">
            <asp:Button ID="UpdateButton" runat="server" Font-Bold="True"
              Font-Names="Copperplate Gothic Bold" Font-Size="X-Large"
Height="47px" onclick="UpdateButton_Click" Text="UPDATE" Width="148px" />
            </td>
          <td class="style3">
            <asp:Button ID="SearchButton" runat="server" Font-Bold="True"
              Font-Names="Copperplate Gothic Bold" Font-Size="X-Large"
Height="47px" onclick="SearchButton_Click" Text="SEARCH" Width="148px" />
          </td>
          <td class="style4">
            <asp:Button ID="DeleteButton" runat="server" Font-Bold="True"
              Font-Names="Copperplate Gothic Bold" Font-Size="X-Large"
Height="47px" onclick="DeleteButton_Click" Text="DELETE" Width="148px" />
          </td>
      </table>
    </div>
    </form>
</body>
</html>
```

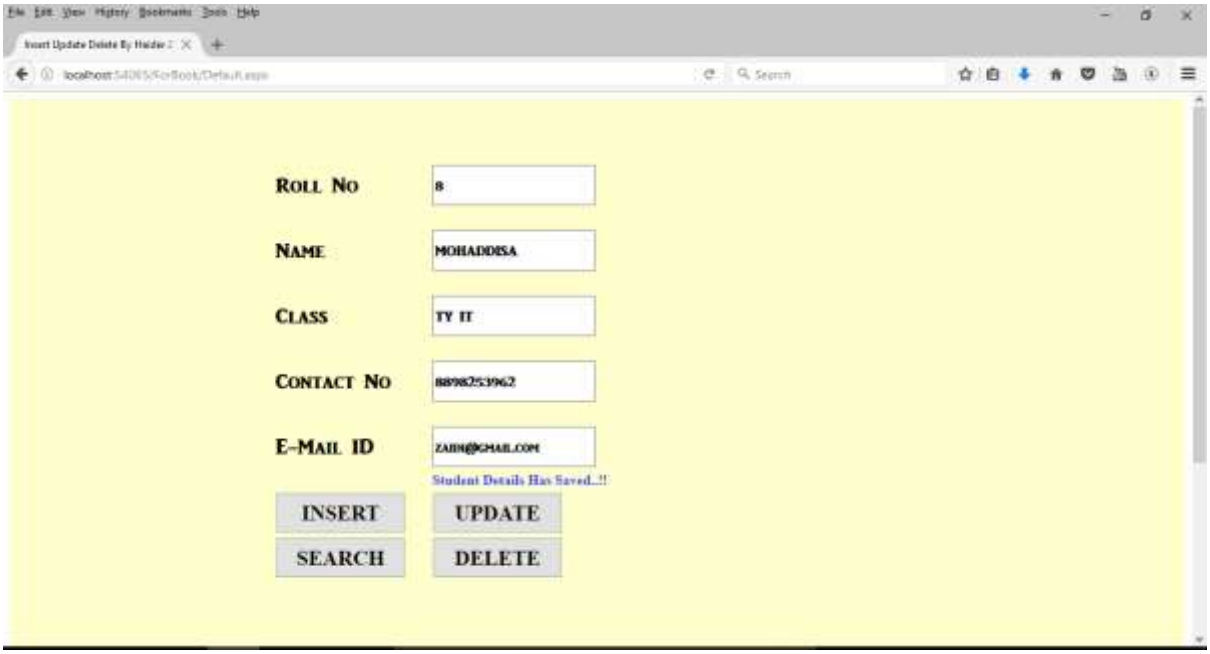
C# Code (Default.aspx.cs):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;          // compulsory for ado.net
using System.Data.SqlClient; // compulsory for ado.net

public partial class _Default : System.Web.UI.Page
{
   SqlConnection con = new SqlConnection("Data Source=hp;Initial Catalog=zaidi;
Integrated Security=True");

   SqlCommand cmd;
   SqlDataReader dr;
   protected void Page_Load(object sender, EventArgs e)
   {

   }
   // Code for Insertion Operation
   protected void InsertButton_Click(object sender, EventArgs e)
   {
      con.Open();
      string s = "insert into student (no , name , class , mobile , email) values ( '" +
TextBox1.Text + "' ,'" + TextBox2.Text + "' ,'" + TextBox3.Text + "' ,'" + TextBox4.Text + "'
,'" + TextBox5.Text + "'   ) ";

      cmd = new SqlCommand(s, con);
      cmd.CommandType = CommandType.Text;

      try
      {
         int result = cmd.ExecuteNonQuery();
         if (result > 0)
         {
            Label6.Text = "Student Details Has Saved..!!";
         }
         else
         {
            Label6.Text = "Failed To Save Student Details ..!!";
         }
      }
      catch(SqlException ex )
      {
         Label6.Text = ex.Message;
      }
      finally
```

```
        {
            con.Close();
        }

        TextBox1.Text = "";
        TextBox2.Text = "";
        TextBox3.Text = "";
        TextBox4.Text = "";
        TextBox5.Text = "";

    }
    // Code for Updation Operation
    protected void UpdateButton_Click(object sender, EventArgs e)
    {

        string s =("Update student set no = '" + TextBox1.Text + "' , name = '" + TextBox2.Text
+ "' , class = '" + TextBox3.Text + "', mobile ='" + TextBox4.Text + "' , email ='" +
TextBox5.Text + "' where no = '"+ TextBox1.Text+"' ");

         con.Open();
        cmd = new SqlCommand(s, con);
        cmd.CommandType = CommandType.Text;
        try
        {
            int result = cmd.ExecuteNonQuery();
            if (result > 0)
            {
                Label6.Text = "Student Details Updated..!!";
            }
            else
            {
                Label6.Text = "Failed To Update Student Details ..!!";
            }
        }
        catch (SqlException ex)
        {
            Label6.Text = ex.Message;
        }
        finally
        {
            con.Close();
        }

        TextBox1.Text = TextBox2.Text = TextBox3.Text = TextBox4.Text = TextBox5.Text =
"";
        }
    // Code for Deletion Operation
    protected void DeleteButton_Click(object sender, EventArgs e)
    {
        string s = ("delete from student where no = '" + TextBox1.Text + "' ");
```

```
    con.Open();
    cmd = new SqlCommand(s, con);
    cmd.CommandType = CommandType.Text;
    try
    {
       int result = cmd.ExecuteNonQuery();
       if (result > 0)
       {
          Label6.Text = "Student Details Deleted..!!";
       }
       else
       {
          Label6.Text = "Failed To Delete Student Details ..!!";
       }
    }
    catch (SqlException ex)
    {
       Label6.Text = ex.Message;
    }
    finally
    {
       con.Close();
    }

TextBox1.Text = TextBox2.Text = TextBox3.Text = TextBox4.Text = TextBox5.Text = "";

  }
  // Code for Searching Operation
  protected void SearchButton_Click(object sender, EventArgs e)
  {
    string s = "select * from student where no = '" + TextBox1.Text + "' ";
     con.Open();
     cmd = new SqlCommand(s, con);
     cmd.CommandType = CommandType.Text;


     try
     {
        dr = cmd.ExecuteReader();
        if (dr.Read())
        {
           TextBox1.Text = dr[0].ToString();
           TextBox2.Text = dr[1].ToString();
           TextBox3.Text = dr[2].ToString();
           TextBox4.Text = dr[3].ToString();
           TextBox5.Text = dr[4].ToString();
        }
        else
        {
```

```
            Label6.Text = "No Record Found ..!!";
        }
        dr.Close();

    }
    catch (SqlException ex)
    {
        Label6.Text = ex.Message;
    }
    finally
    {
        con.Close();
    }

  }
}
```



11.4.2 Disconnected Architecture

Disconnected is the main feature of the .NET framework. ADO.NET contains various classes that support this architecture. The .NET application does not always stay connected with the database. The classes are designed in a way that they automatically open and close the connection. The data is stored client-side and is updated in the database whenever required. The ADO.NET Discon nected architecture considers primarily the following types of objects:

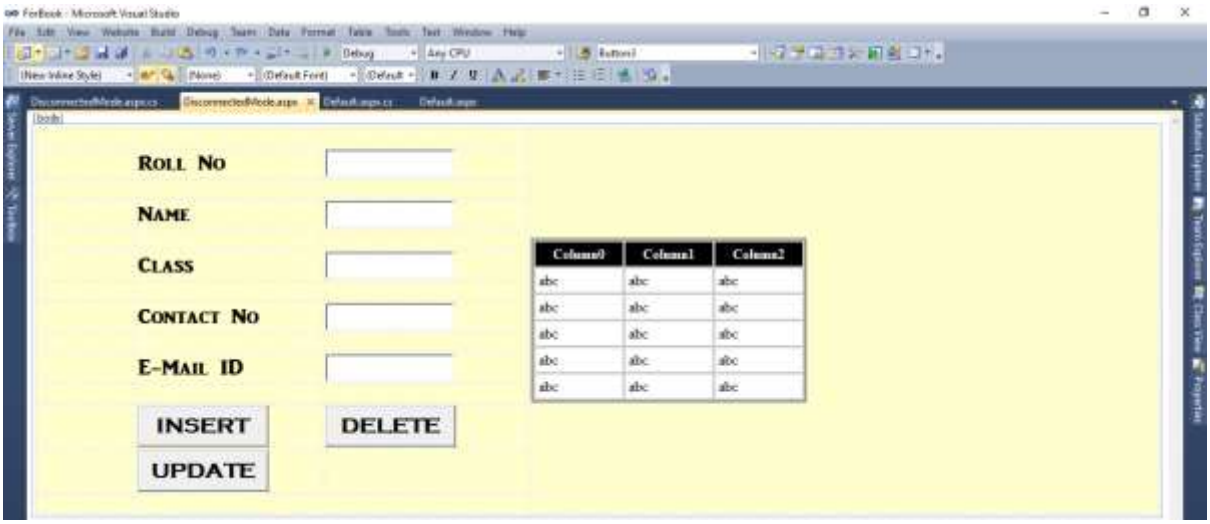
- DataSet ds;
- SqlDataAdapter da;
- SqlConnection con;
- SqlCommandBuilder bldr;

11.4.2.1 Disconnected Data Access

- The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture.
- When you use disconnected data access, you keep a copy of your data in memory using the DataSet. You connect to the database just long enough to fetch your data and dump it into the DataSet, and then you disconnect immediately.
- Method of retrieving a record set from the database and storing it giving the ability to do many CRUD (Create, Read, Update and Delete) operations on the data in memory, then it can be re-synchronized with the database when reconnecting.

11.4.2.2 Inserting, Updating, and Deleting in disconnected mode

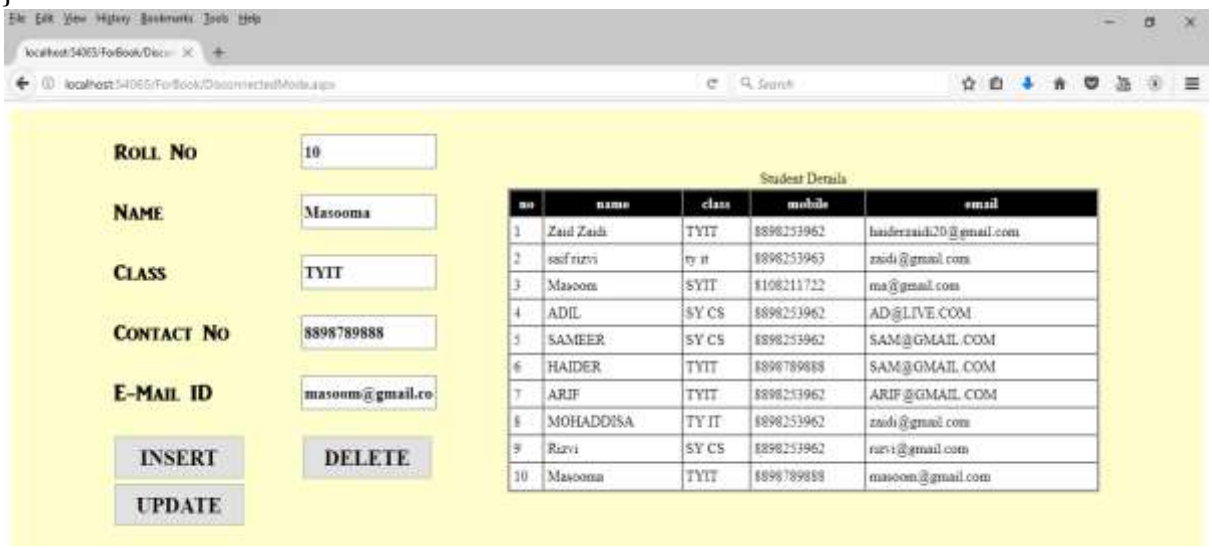
- We are using same database which is used in connected mode of ADO.NET.
- Create a web page same as connected mode.
- Drag and drop a GridView control on the web page.
- After designing page will be like this.



C# Code (Default.aspx.cs):

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.SqlClient;

public partial class DisconnectedMode : System.Web.UI.Page
{
        SqlConnection con = new SqlConnection("Data Source=hp;Initial
Catalog=zaidi;Integrated Security=True");
   public DataTable dt;
   SqlDataAdapter da;
   SqlCommandBuilder cmd;

   protected void Page_Load(object sender, EventArgs e)
   {

      da = new SqlDataAdapter("select * from student", con);

      DataSet ds = new DataSet();
      da.Fill(ds);
      GridView1.DataSource = ds.Tables[0];
      GridView1.DataBind();
   }
   protected void InsertButton_Click(object sender, EventArgs e)
   {
      da = new SqlDataAdapter("select * from student", con);
      da.InsertCommand = new SqlCommand("insert into student
values(@p1,@p2,@p3,@p4,@p5)", con);

      SqlCommand cmd = da.InsertCommand;

      cmd.Parameters.AddWithValue("@p1", TextBox1.Text);
      cmd.Parameters.AddWithValue("@p2", TextBox2.Text);
      cmd.Parameters.AddWithValue("@p3", TextBox3.Text);
      cmd.Parameters.AddWithValue("@p4", TextBox4.Text);
      cmd.Parameters.AddWithValue("@p5", TextBox5.Text);
      con.Open();
      cmd.ExecuteNonQuery();
      DataSet ds = new DataSet();
      da.Fill(ds);
      GridView1.DataSource = ds.Tables[0];
      GridView1.DataBind();
      con.Close();
   }
   protected void DeleteButton_Click(object sender, EventArgs e)
   {
```

```
        da.DeleteCommand = new SqlCommand("delete from student where no = '" +
TextBox1.Text + "' ",con);
        SqlCommand cmd = da.DeleteCommand;
        con.Open();
        cmd.ExecuteNonQuery();
        DataSet ds = new DataSet();
        da.Fill(ds);
        GridView1.DataSource = ds.Tables[0];
        GridView1.DataBind();
        con.Close();
    }
    protected void UpdateButton_Click(object sender, EventArgs e)
    {       string s = ("Update student set no = '" + TextBox1.Text + "' , name = '" +
TextBox2.Text + "' , class = '" + TextBox3.Text + "', mobile ='" + TextBox4.Text + "' , email
='" + TextBox5.Text + "' where no = '" + TextBox1.Text + "' ");

        da.UpdateCommand = new SqlCommand(s, con);
        SqlCommand cmd = da.UpdateCommand;
        con.Open();
        cmd.ExecuteNonQuery();
        DataSet ds = new DataSet();
        da.Fill(ds);
        GridView1.DataSource = ds.Tables[0];
        GridView1.DataBind();
        con.Close();

    }
}
```



11.5 DIFFERENCE BETWEEN CONNECTED AND DISCONNECTED ARCHITECTURE IN ASP.NET

Sr.No	Connected	Disconnected
1	It is connection oriented.	It is disconnection oriented.
2	DataReader is used for retrieving data from the database	DataSet is used for retrieving data from the database
3	Connected methods gives faster performance	Disconnected get low in speed and performance.
4	In Connected .net runtime creates an instance of the DataTable to hold data. Connected can hold the data of single table.	In disconnected-data can be accessed from multiple tables in a dataset. Disconnected can hold multiple tables of data.
5	In connected you need to use a read only forward only data reader	In disconnected you cannot
6	Data Reader can't persist the data	Data Set can persist the data
7	It is Read only, we can't update the data.	We can update data
8	All the operations can be performed as the data is accessed in the database.	All the operations can be performed with the data once retrieved.

11.6 SUMMARY

This unit gives an overview of ADO.NET architecture and its components. Further it discusses operation of ADO.NET in connected and disconnected mode.

11.7 EXERCISE

1. What is ADO.NET? Explain ADO.NET architecture.
2. Write a note on sqlConnection and sqlCommand Class.
3. Write a note on Data Reader and Data Adapter.
4. Explain connected and disconnected mode of ADO.NET.
5. What is the difference between connected and disconnected mode of ADO.NET.

REFERENCE

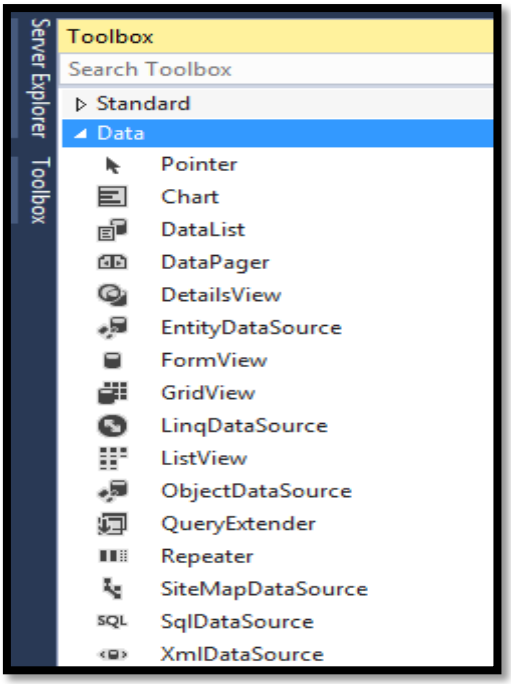
1. ADO.NET: The Complete Reference
2. Beginning ASP.NET 4 by Imar Spaanjaars
3. Database Programming with Visual Basic .Net and ADO.NET by F Scott Barker, Publisher: Pearson Education
4. https://www.c-sharpcorner.com/
5. https://www.aspdotnet-suresh.com/
6. https://stackoverflow.com/
7. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview

DATA CONTROL IN ASP.NET

Content

12.0 ASP.NET DATA CONTROL

ASP.NET provides a wide variety of rich controls that can be bound to data. Under the Data tab of the Visual Studio Toolbox, you can get several controls under the Data tab that could be used to display data from a data source, like a database or XML file.



12.1 ASP.NET DATA SOURCE CONTROL

A data source control interacts with the data-bound controls and hides the complex data binding processes. These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

Each data source control wraps a particular data provider-relational databases, XML documents, or custom classes and helps in:

- Managing connection
- Selecting data
- Managing presentation aspects like paging, caching, etc.
- Manipulating data

Data source controls provide a consistent and extensible method for declaratively accessing data from web pages. Data source controls available with ASP.NET 4.0 are as follows:

1. **<asp:SqlDataSource>:** This data source control is used to work with SQL Server, OLE DB, Open DataBase Connectivity (ODBC), and Oracle databases. Using this control, we can also select, update, delete, and insert data using SQL commands.
2. **<asp:ObjectDataSource>:** N-tier methodology allows you to create web applications that are not only scalable but also easier to maintain. N-tier principle also enables clean separation, thereby allowing you to easily add new functionalities. In an n-tier application, the middle-tier objects may return complex objects that you have to process in your ASP.NET presentation layer. Keeping this requirement in mind, Microsoft has created this new control that allows you to seamlessly integrate the data returned from the middle-layer objects with the ASP.NET presentation layer.
3. **<asp:AccessDataSource>:** This is very similar to the SqlDataSource control, except for the difference that it is designed to work with Access databases.
4. **<asp:XmlDataSource>:** Allows you to bind to XML data, which can come from a variety of sources, such as an external XML file, a DataSet object, and so on. Once the XML data is bound to the XmlDataSource control, this control can then act as a source of data for data-bound controls such as TreeView and Menu.
5. **<asp:SiteMapDataSource>:** Provides a site navigation framework that makes the creation of a site navigation system a breezy experience. Accomplishing this requires the use of a new XML file named web.sitemap that lays out the pages of the site in a hierarchical XML structure. Once you have the site hierarchy in the web.sitemap file, you can then data-bind the SiteMap DataSource control with the web.sitemap file. Then the contents of the SiteMapDataSource control can be bound to data-aware controls such as TreeView, Menu, and so on.

12.1.1 GRIDVIEW CONTROL

- The GridView control is one of the most powerful user interface controls available in ASP.NET 4.
- It was introduced with ASP.NET 2.0. The GridView control is used to display the values of a data source in a table.
- It provides many options that let you customize its appearance and behavior.
- The GridView control displays data provided by a data source in a row and column format.
- Each column represents a field where each row represents a record. It can also display empty data. The GridView control provides many built-in capabilities that allow the user to sort, update, delete, select and page through items in the control.
- The GridView control renders its data as an HTML table with one Tr element for each row in the data source, and one Td element for each column in the data source.
- Most of the .aspx code for a GridView control is created automatically by Visual Studio when you drag the control from the Toolbox onto the form and when you use the configuration wizard to configure the data source.

The GridView control supports the following features:

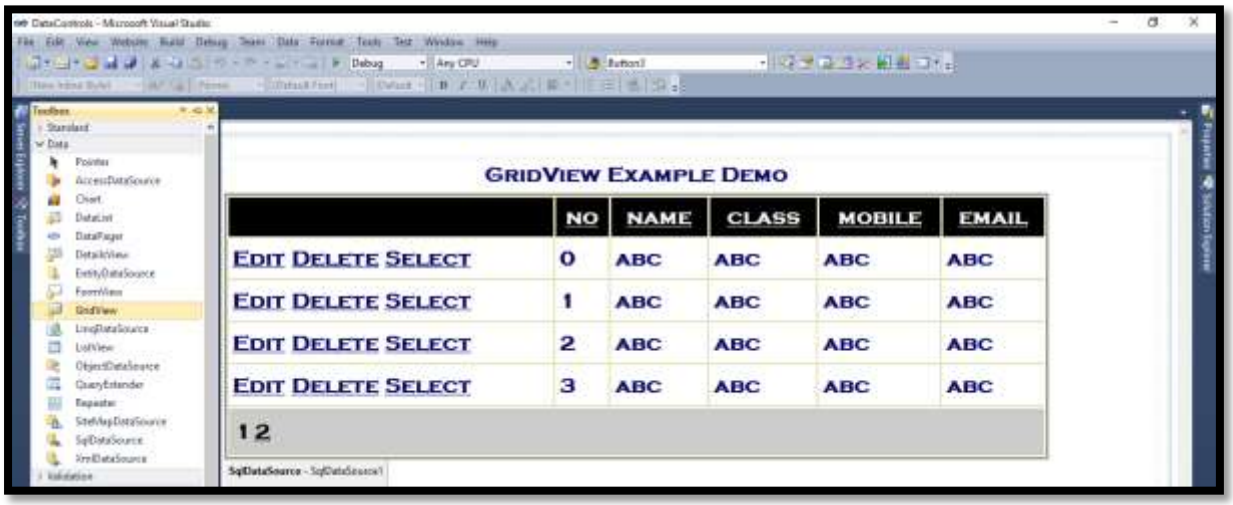
- Improved data source binding capabilities.
- Tabular rendering – displays data as a table.
- Item as row.
- Built-in sorting capability.
- Built-in select, edit and delete capabilities.
- Built-in paging capability.
- Built-in row selection capability.
- Multiple key fields.
- Programmatic access to the GridView object model to dynamically set properties, handle events and so on.
- Richer design-time capabilities.
- Control over Alternate item, Header, Footer, Colors, font, borders, and so on.

Basic attributes of the GridView control

Attribute	Description
ID	The ID of the control
Runat	Must specify “Server”
DataSourceID	The ID of the data source to bind to.
DataKeyNames	The name of the primary key fields separated by commas
AutoGenrateColumne	Specifies whether the control's columns should be automatically generated.
SelectedIndex	Specifies the row to be initially selected
AllowPaging	true/false. Indicate whether the control should support paging.
AllowSorting	true/false. Indicate whether the control should support sorting.
Caption	Gets or sets the caption of the GridView.
CellPadding	Indicates the space in pixel between the cells and the border of

	the GridView.
CellSpacing	Indicates the space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GrdiLines should appear or not, if yes Horizontal, Vertical or Both.

Example: Demo of GridView

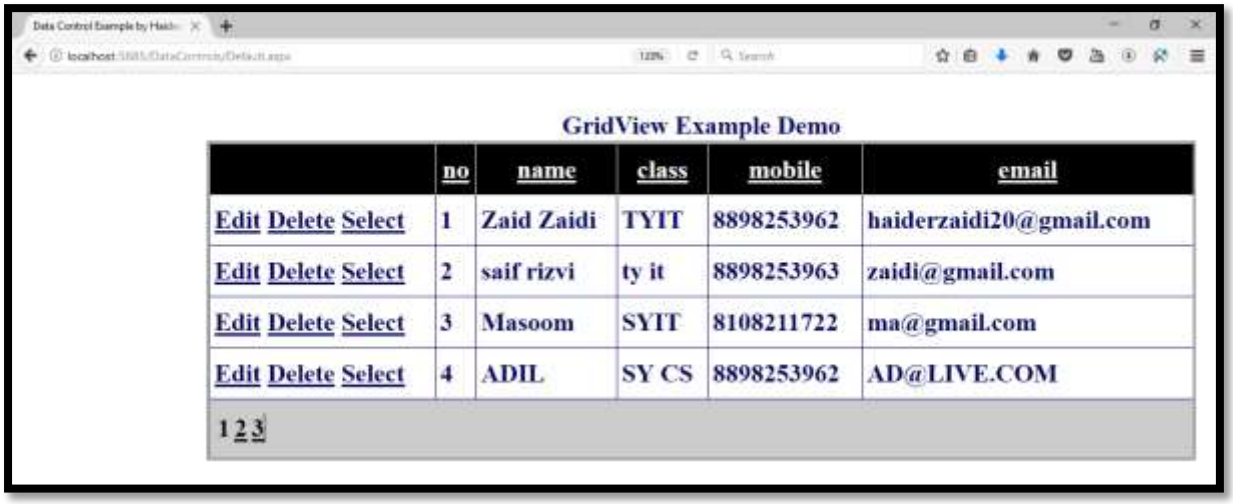


Default.aspx source code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Data Control Example by Haider Sir</title>
</head>
<body>
  <form id="form1" runat="server">

            <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
              AllowSorting="True" AutoGenerateColumns="False"
BackColor="#FFFFCC"
              BorderColor="#999999" BorderStyle="Solid" BorderWidth="3px"
              Caption="GridView Example Demo" CaptionAlign="Top" CellPadding="4"
              CellSpacing="2" DataKeyNames="no" DataSourceID="SqlDataSource1"
              Font-Bold="True" Font-Names="Copperplate Gothic Bold" Font-Size="X-
Large"
              ForeColor="#000066" Height="302px" PageSize="4" Width="935px">
              <Columns>
          <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
           ShowSelectButton="True" />
                <asp:BoundField DataField="no" HeaderText="no" ReadOnly="True"
                  SortExpression="no" />
                <asp:BoundField DataField="name" HeaderText="name"
SortExpression="name" />
                <asp:BoundField DataField="class" HeaderText="class"
SortExpression="class" />
```

```
                    <asp:BoundField DataField="mobile" HeaderText="mobile"
                        SortExpression="mobile" />
                    <asp:BoundField DataField="email" HeaderText="email"
SortExpression="email" />
                </Columns>
                <FooterStyle BackColor="#CCCCCC" />
                <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
                <PagerStyle BackColor="#CCCCCC" ForeColor="Black"
HorizontalAlign="Left" />
                <RowStyle BackColor="White" />
                <SelectedRowStyle BackColor="#000099" Font-Bold="True"
ForeColor="White" />
                <SortedAscendingCellStyle BackColor="#F1F1F1" />
                <SortedAscendingHeaderStyle BackColor="#808080" />
                <SortedDescendingCellStyle BackColor="#CAC9C9" />
                <SortedDescendingHeaderStyle BackColor="#383838" />
            </asp:GridView>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConflictDetection="CompareAllValues"
                ConnectionString="<%$ ConnectionStrings:zaidiConnectionString %>"
                DeleteCommand="DELETE FROM [student] WHERE [no] = @original_no
AND (([name] = @original_name) OR ([name] IS NULL AND @original_name IS NULL))
AND (([class] = @original_class) OR ([class] IS NULL AND @original_class IS NULL))
AND (([mobile] = @original_mobile) OR ([mobile] IS NULL AND @original_mobile IS
NULL)) AND (([email] = @original_email) OR ([email] IS NULL AND @original_email IS
NULL))"
                InsertCommand="INSERT INTO [student] ([no], [name], [class], [mobile],
[email]) VALUES (@no, @name, @class, @mobile, @email)"
                OldValuesParameterFormatString="original_{0}"
                SelectCommand="SELECT * FROM [student]"
                UpdateCommand="UPDATE [student] SET [name] = @name, [class] =
@class, [mobile] = @mobile, [email] = @email WHERE [no] = @original_no AND
(([name] = @original_name) OR ([name] IS NULL AND @original_name IS NULL)) AND
(([class] = @original_class) OR ([class] IS NULL AND @original_class IS NULL)) AND
(([mobile] = @original_mobile) OR ([mobile] IS NULL AND @original_mobile IS NULL))
AND (([email] = @original_email) OR ([email] IS NULL AND @original_email IS
NULL))">
                <DeleteParameters>
                    <asp:Parameter Name="original_no" Type="Int32" />
                    <asp:Parameter Name="original_name" Type="String" />
                    <asp:Parameter Name="original_class" Type="String" />
                    <asp:Parameter Name="original_mobile" Type="String" />
                    <asp:Parameter Name="original_email" Type="String" />
                </DeleteParameters>
                <InsertParameters>
                    <asp:Parameter Name="no" Type="Int32" />
                    <asp:Parameter Name="name" Type="String" />
                    <asp:Parameter Name="class" Type="String" />
                    <asp:Parameter Name="mobile" Type="String" />
                    <asp:Parameter Name="email" Type="String" />
```

```
            </InsertParameters>
            <UpdateParameters>
              <asp:Parameter Name="name" Type="String" />
              <asp:Parameter Name="class" Type="String" />
              <asp:Parameter Name="mobile" Type="String" />
              <asp:Parameter Name="email" Type="String" />
              <asp:Parameter Name="original_no" Type="Int32" />
              <asp:Parameter Name="original_name" Type="String" />
              <asp:Parameter Name="original_class" Type="String" />
              <asp:Parameter Name="original_mobile" Type="String" />
              <asp:Parameter Name="original_email" Type="String" />
            </UpdateParameters>
          </asp:SqlDataSource>
        </td>

    </table>

  </div>
  </form>
</body>
</html>
```



GridView Example Demo

	no	name	class	mobile	email
Edit Delete Select	1	Zaid Zaidi	TYIT	8898253962	haiderzaidi20@gmail.com
Edit Delete Select	2	saif rizvi	ty it	8898253963	zaidi@gmail.com
Edit Delete Select	3	Masoom	SYIT	8108211722	ma@gmail.com
Edit Delete Select	4	ADIL	SY CS	8898253962	AD@LIVE.COM

1 2 3

12.1.2 DETAILSVIEW CONTROL

- The DetailsView control is designed to display the data for a single item of a data source.
- To use this control effectively, you must provide some way for the user to select which data item to display.
- The most common way to do that is to use the DetailsView control in combination with another control such as a GridView control or a drop-down list.
- A DetailsView control can be displayed in one of three modes.
  1. In Read-only mode, the data for the current data source row is displayed but can't be modified.
  2. In Edit mode, the user can modify the data for the current row.
  3. In Insert mode, the user can enter data that will be inserted into the data source as a new row.

The DetailsView control supports the following features:

- Tabular rendering.

- Supports column layout, by default two columns at a time.
- Optional support for paging and navigation.
- Built-in support for data grouping.
- Built-in support for edit, insert and delete capabilities.

DetailsView control attributes

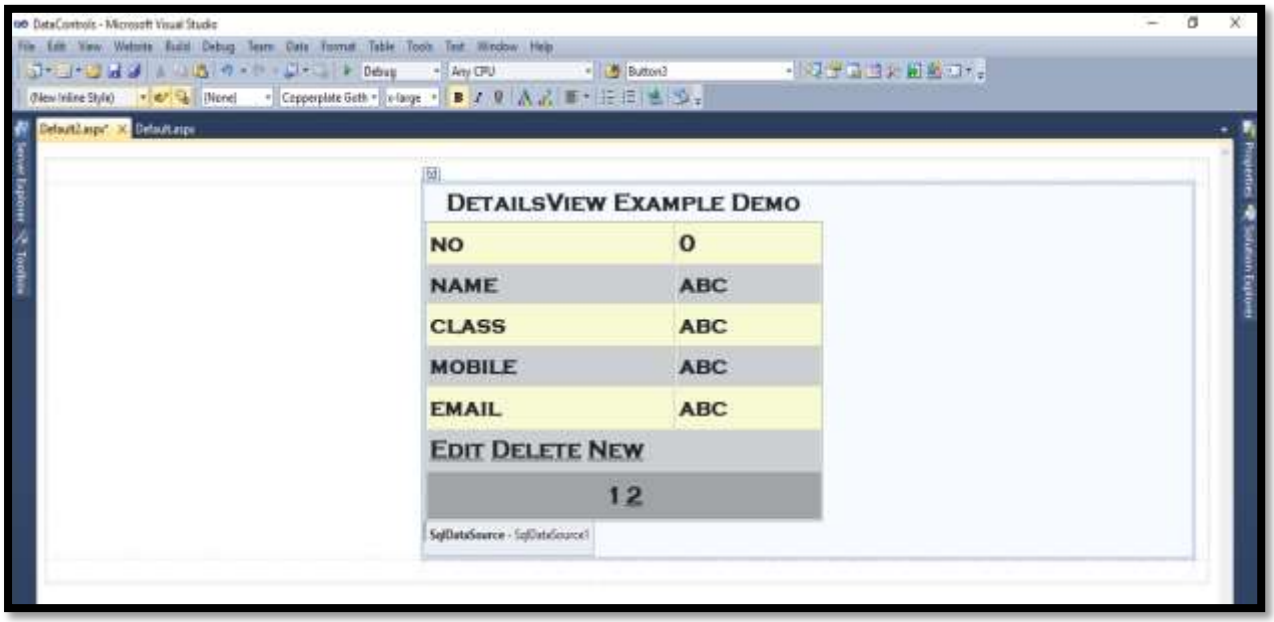
Attribute	Description
ID	The ID of the control
Runat	Must specify “Server”
DataSourceID	The ID of the data source to bind to.
DataKeyNames	A list of field names that form the primary key for the data source.
AutoGenrateColumne	If True, a row is automatically generated for each field in the data source. If False, you must define the rows in the Fields element.
DefaultMode	Sets the initial mode of the DetailsView control. Valid options are Edit, Insert, or ReadOnly.
AllowPaging	Set to True to allow paging.

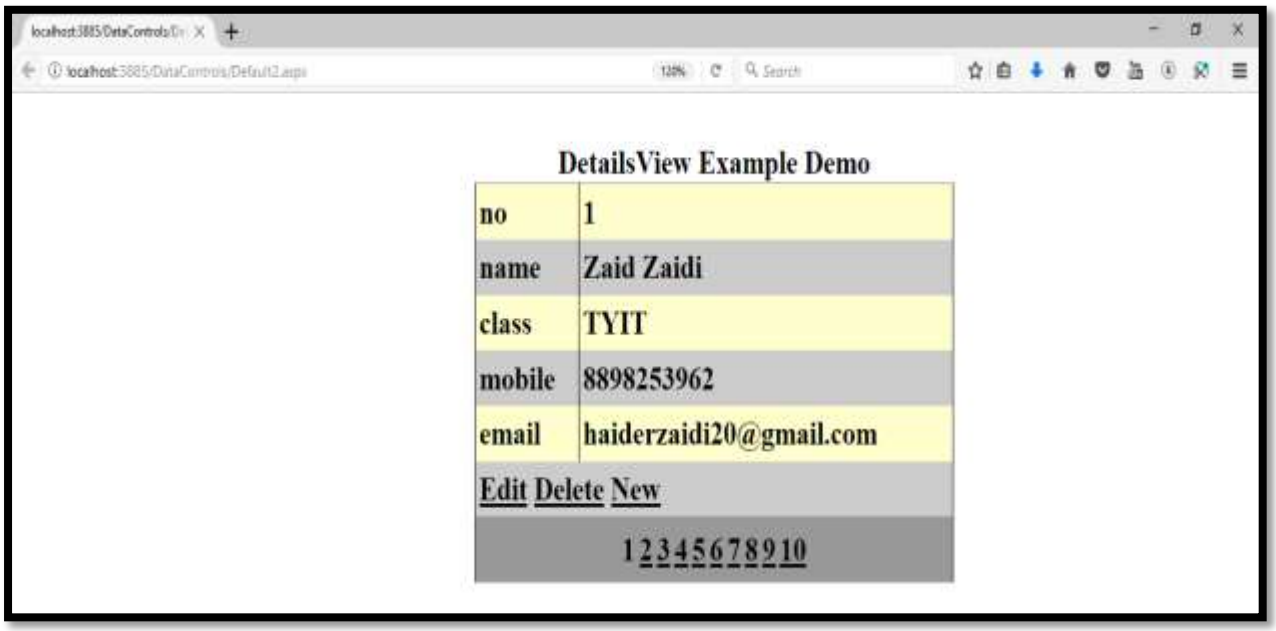
Default2.aspx source code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
        <title>Data Control Example by Haider Sir</title>
</head>
<body>
   <form id="form1" runat="server">
   <div><table class="style1">
                    <asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="True"
                AutoGenerateRows="False" BackColor="#FFFFCC" BorderColor="#999999"
                BorderStyle="Solid" BorderWidth="1px" Caption="DetailsView Example
Demo" CellPadding="3" CellSpacing="1" DataKeyNames="no"
DataSourceID="SqlDataSource1"
                Font-Bold="True" Font-Names="Copperplate Gothic Bold" Font-Size="X-
Large"
                ForeColor="Black" GridLines="Vertical" Height="287px" Width="436px">
                <AlternatingRowStyle BackColor="#CCCCCC" />
                <EditRowStyle BackColor="#000099" Font-Bold="True" ForeColor="White"
/>
                <Fields>
                   <asp:BoundField DataField="no" HeaderText="no" ReadOnly="True"
                      SortExpression="no" />
                   <asp:BoundField DataField="name" HeaderText="name"
SortExpression="name" />
                   <asp:BoundField DataField="class" HeaderText="class"
SortExpression="class" />
                   <asp:BoundField DataField="mobile" HeaderText="mobile"
                      SortExpression="mobile" />
```

```
                <asp:BoundField DataField="email" HeaderText="email"
SortExpression="email" />
                <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
                  ShowInsertButton="True" />
              </Fields>
              <FooterStyle BackColor="#CCCCCC" />
              <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
              <PagerStyle BackColor="#999999" ForeColor="Black"
HorizontalAlign="Center" />
            </asp:DetailsView>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
              ConflictDetection="CompareAllValues"
              ConnectionString="<%$ ConnectionStrings:zaidiConnectionString2 %>"
              DeleteCommand="DELETE FROM [student] WHERE [no] = @original_no
AND (([name] = @original_name) OR ([name] IS NULL AND @original_name IS NULL))
AND (([class] = @original_class) OR ([class] IS NULL AND @original_class IS NULL))
AND (([mobile] = @original_mobile) OR ([mobile] IS NULL AND @original_mobile IS
NULL)) AND (([email] = @original_email) OR ([email] IS NULL AND @original_email IS
NULL))"
              InsertCommand="INSERT INTO [student] ([no], [name], [class], [mobile],
[email]) VALUES (@no, @name, @class, @mobile, @email)"
              OldValuesParameterFormatString="original_{0}"
              SelectCommand="SELECT * FROM [student]"
              UpdateCommand="UPDATE [student] SET [name] = @name, [class] =
@class, [mobile] = @mobile, [email] = @email WHERE [no] = @original_no AND
(([name] = @original_name) OR ([name] IS NULL AND @original_name IS NULL)) AND
(([class] = @original_class) OR ([class] IS NULL AND @original_class IS NULL)) AND
(([mobile] = @original_mobile) OR ([mobile] IS NULL AND @original_mobile IS NULL))
AND (([email] = @original_email) OR ([email] IS NULL AND @original_email IS
NULL))">
              <DeleteParameters>
                <asp:Parameter Name="original_no" Type="Int32" />
                <asp:Parameter Name="original_name" Type="String" />
                <asp:Parameter Name="original_class" Type="String" />
                <asp:Parameter Name="original_mobile" Type="String" />
                <asp:Parameter Name="original_email" Type="String" />
              </DeleteParameters>
              <InsertParameters>
                <asp:Parameter Name="no" Type="Int32" />
                <asp:Parameter Name="name" Type="String" />
                <asp:Parameter Name="class" Type="String" />
                <asp:Parameter Name="mobile" Type="String" />
                <asp:Parameter Name="email" Type="String" />
              </InsertParameters>
              <UpdateParameters>
                <asp:Parameter Name="name" Type="String" />
                <asp:Parameter Name="class" Type="String" />
                <asp:Parameter Name="mobile" Type="String" />
                <asp:Parameter Name="email" Type="String" />
                <asp:Parameter Name="original_no" Type="Int32" />
```

```
                    <asp:Parameter Name="original_name" Type="String" />
                    <asp:Parameter Name="original_class" Type="String" />
                    <asp:Parameter Name="original_mobile" Type="String" />
                    <asp:Parameter Name="original_email" Type="String" />
                </UpdateParameters>
            </asp:SqlDataSource>
        </td>
    </table>
  </div>
  </form>
</body>
</html>
```



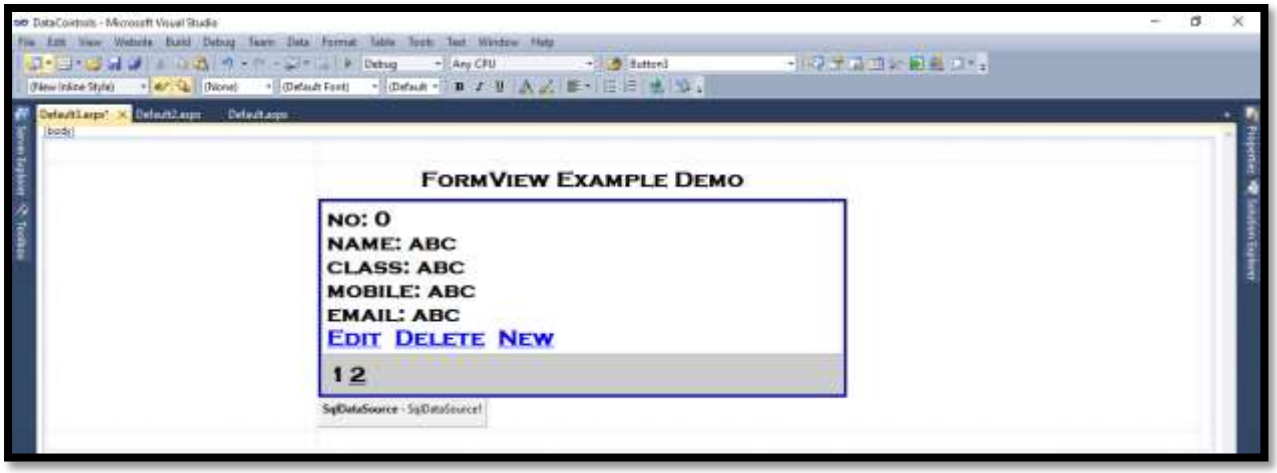


12.1.3 FORMVIEW CONTROL

- Besides the DetailsView control, ASP.NET also provides a FormView control. Like the DetailsView control, the FormView control is designed to display data for a single item from a data source.
- A FormView control is similar to a DetailsView control, but its templates give you more control over how its data is displayed. To accomplish that, all the columns in the data source can be laid out within a single template.
- The FormView control renders a single data item at a time from a data source, even if its data source exposes a multiple records data item from a data source. It allows for a more flexible layout when displaying a single record. The FormView control renders all fields of a single record in a single table row. In contrast, the FormView control does not specify a pre-defined layout for displaying a record. Instead, you create templates that contain controls to display individual fields from the record.
- After you create a FormView control and assign a data source to it, you can edit the control's templates so the data is displayed the way you want.

The FormView control supports the following features:

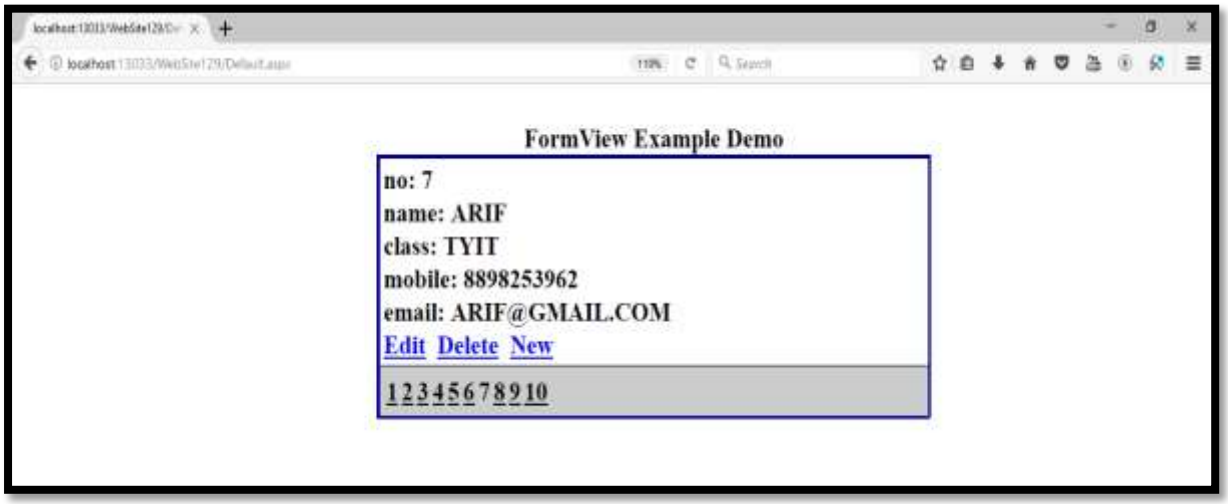
- Template driven
- Supports column layout
- Built-in support for paging and grouping
- Built-in support for insert, edit and delete capabilities




```
<asp:FormView ID="FormView1" runat="server" AllowPaging="True"
                BackColor="#CCCCCC" BorderColor="#000099" BorderStyle="Solid"
BorderWidth="3px"
              Caption="FormView Example Demo" CaptionAlign="Top" CellPadding="4"
              CellSpacing="2" DataKeyNames="no" DataSourceID="SqlDataSource1"
                Font-Bold="True" Font-Names="Copperplate Gothic Bold" Font-Size="X-
Large"
                ForeColor="Black" GridLines="Both" Width="578px">
                <EditItemTemplate>
                   no:
                   <asp:Label ID="noLabel1" runat="server" Text='<%# Eval("no") %>' />
                   <br />
                   name:
                   <asp:TextBox ID="nameTextBox" runat="server" Text='<%#
Bind("name") %>' />
```

```
                    <br />
                    class:
                    <asp:TextBox ID="classTextBox" runat="server" Text='<%# Bind("class")
%>' />
                    <br />
                    mobile:
                    <asp:TextBox ID="mobileTextBox" runat="server" Text='<%#
Bind("mobile") %>' />
                    <br />
                    email:
                    <asp:TextBox ID="emailTextBox" runat="server" Text='<%#
Bind("email") %>' />
                    <br />
                    <asp:LinkButton ID="UpdateButton" runat="server"
CausesValidation="True"
                        CommandName="Update" Text="Update" />
                     <asp:LinkButton ID="UpdateCancelButton" runat="server"
                        CausesValidation="False" CommandName="Cancel" Text="Cancel" />
                </EditItemTemplate>
                <EditRowStyle BackColor="#000099" Font-Bold="True" ForeColor="White"
/>
                <FooterStyle BackColor="#CCCCCC" />
                <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
                <InsertItemTemplate>
                    no:
                    <asp:TextBox ID="noTextBox" runat="server" Text='<%# Bind("no") %>'
/>
                    <br />
                    name:
                    <asp:TextBox ID="nameTextBox" runat="server" Text='<%#
Bind("name") %>' />
                    <br />
                    class:
                    <asp:TextBox ID="classTextBox" runat="server" Text='<%# Bind("class")
%>' />
                    <br />
                    mobile:
                    <asp:TextBox ID="mobileTextBox" runat="server" Text='<%#
Bind("mobile") %>' />
                    <br />
                    email:
                    <asp:TextBox ID="emailTextBox" runat="server" Text='<%#
Bind("email") %>' />
                    <br />
                    <asp:LinkButton ID="InsertButton" runat="server"
CausesValidation="True"
                        CommandName="Insert" Text="Insert" />
                     <asp:LinkButton ID="InsertCancelButton" runat="server"
                        CausesValidation="False" CommandName="Cancel" Text="Cancel" />
                </InsertItemTemplate>
```

```
                <ItemTemplate>
                    no:
                    <asp:Label ID="noLabel" runat="server" Text='<%# Eval("no") %>' />
                    <br />
                    name:
                    <asp:Label ID="nameLabel" runat="server" Text='<%# Bind("name") %>'
/>
                    <br />
                    class:
                    <asp:Label ID="classLabel" runat="server" Text='<%# Bind("class") %>'
/>
                    <br />
                    mobile:
                    <asp:Label ID="mobileLabel" runat="server" Text='<%# Bind("mobile")
%>' />
                    <br />
                    email:
                    <asp:Label ID="emailLabel" runat="server" Text='<%# Bind("email") %>'
/>
                    <br />
                    <asp:LinkButton ID="EditButton" runat="server"
CausesValidation="False"
                        CommandName="Edit" Text="Edit" />
                     <asp:LinkButton ID="DeleteButton" runat="server"
CausesValidation="False"
                        CommandName="Delete" Text="Delete" />
                     <asp:LinkButton ID="NewButton" runat="server"
CausesValidation="False"
                        CommandName="New" Text="New" />
                </ItemTemplate>
                <PagerStyle BackColor="#CCCCCC" ForeColor="Black"
HorizontalAlign="Left" />
                <RowStyle BackColor="White" />
            </asp:FormView>
```



How the FormView control differs from the DetailsView control

- The DetailsView control can be easier to work with, but the FormView control provides more formatting and layout options.
- The DetailsView control can use BoundField elements or TemplateField elements with templates that use data binding expressions to define bound data fields. The FormView control can use only templates with data binding expressions to display bound data.
- The DetailsView control renders each field as a table row, but the FormView control renders all the fields in a template as a single table row.
- When you bind a FormView control to a data source, the Web Forms Designer generates an Item template that includes heading text and a bound label for each column in the data source.
- The Item template is rendered whenever the FormView control is displayed in ReadOnly mode.
- The Item template uses the Eval and Bind methods to create binding expressions for the columns in the data source.
- If the data source includes Update, Delete, and Insert commands, the generated Item template will include Edit, Delete, and New buttons.
- The Web Forms Designer also generates an Editltem template and an Insertltem template, even if the data source doesn't include an Update or Insert command. For more information, see the next figure.
- You can modify a generated template so you can use CSS to control the format and layout of the data that's rendered for that template.

12.2 DEPLOYING THE WEB SITE

Once you have put in all the hard work of creating a website, you need to get it on the web so people can navigate to it and access its content. This process is called deployment.

In simple words, “deployment” simply means getting your website files onto the server.

There are two categories of ASP.NET deployment:

- **Local deployment** : In this case, the entire application is contained within a virtual directory and all the contents and assemblies are contained within it and available to the application.
- **Global deployment** : In this case, assemblies are available to every application running on the server.

There are different techniques used for deployment, however, we will discuss the following most common and easiest ways of deployment:

- XCOPY deployment
- Copying a Website
- Creating a set up project

XCOPY Deployment

XCOPY deployment means making recursive copies of all the files to the target folder on the target machine. You can use any of the commonly used techniques:

- FTP transfer
- Using Server management tools that provide replication on a remote site
- MSI installer application

XCOPY deployment simply copies the application file to the production server and sets a virtual directory there. You need to set a virtual directory using the Internet Information Manager Microsoft Management Console (MMC snap-in).

Copying a Website

The Copy Web Site option is available in Visual Studio. It is available from the Website -> Copy Web Site menu option. This menu item allows copying the current web site to another local or remote location. It is a sort of integrated FTP tool.

Using this option, you connect to the target destination, select the desired copy mode:

- Overwrite
- Source to Target Files
- Sync UP Source And Target Projects

Then proceed with copying the files physically. Unlike the XCOPY deployment, this process of deployment is done from Visual Studio environment. However, there are following problems with both the above deployment methods:

- You pass on your source code.
- There is no pre-compilation and related error checking for the files.
- The initial page load will be slow.

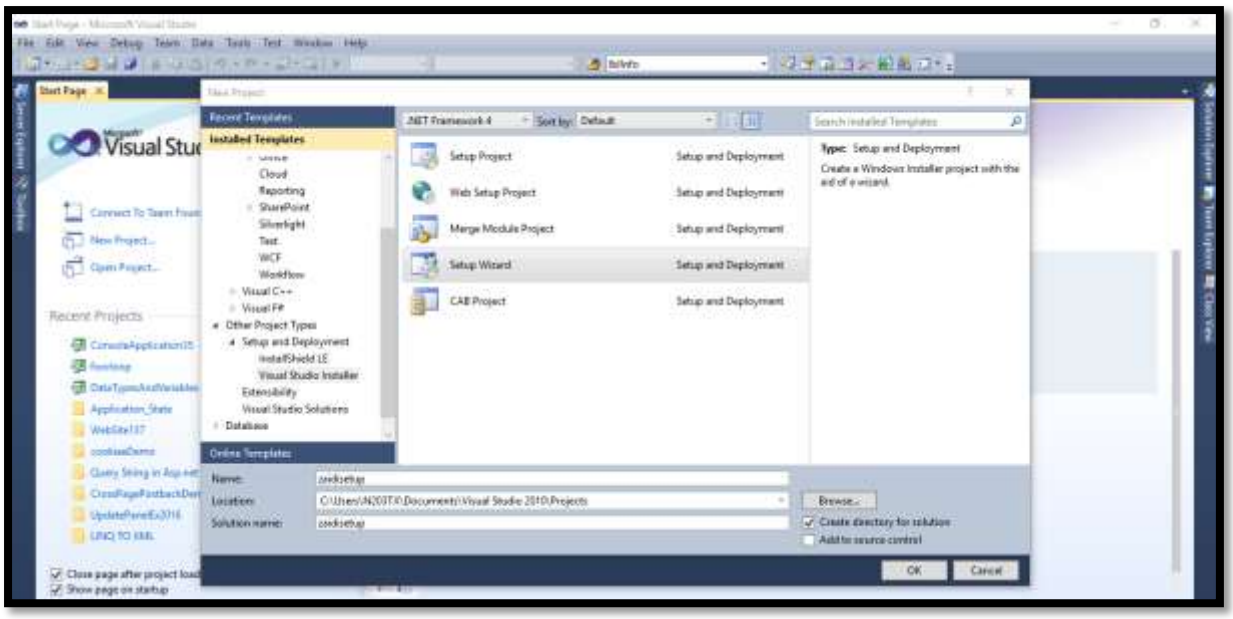
Creating a Setup Project

In this method, you use Windows Installer and package your web applications so it is ready to deploy on the production server. Visual Studio allows you to build deployment packages. Let us test this on one of our existing project, say the data binding project.

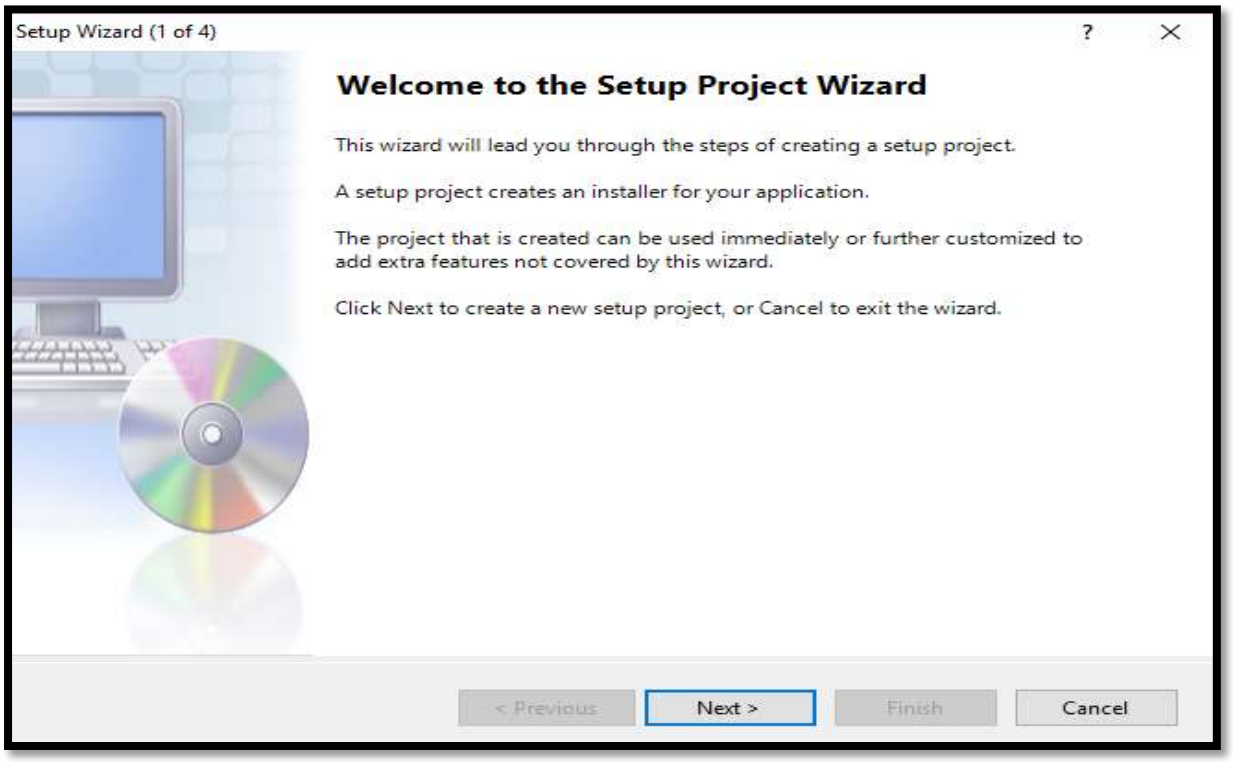
Open the project and take the following steps:

Step (1) : Select File -> Add -> New Project with the website root directory highlighted in the Solution Explorer.

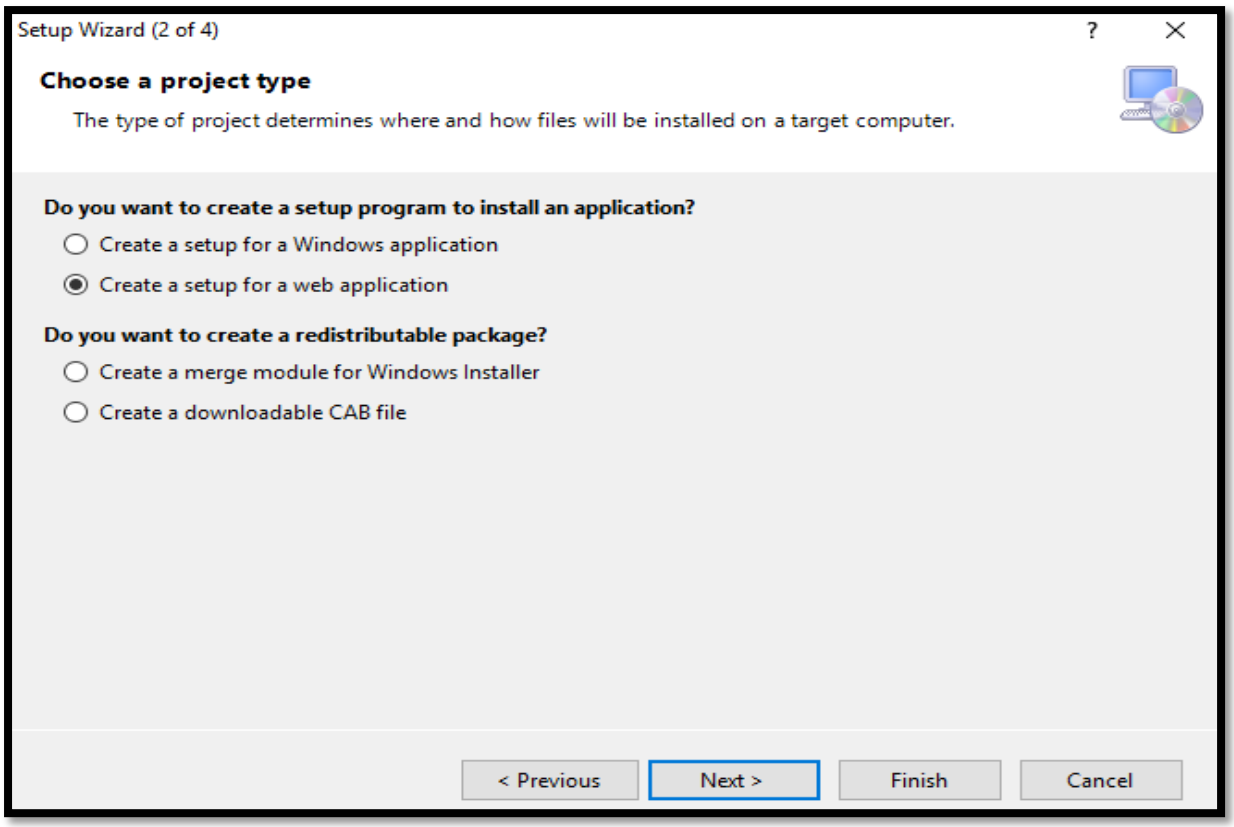
Step (2) : Select Setup and Deployment, under Other Project Types. Select Setup Wizard.



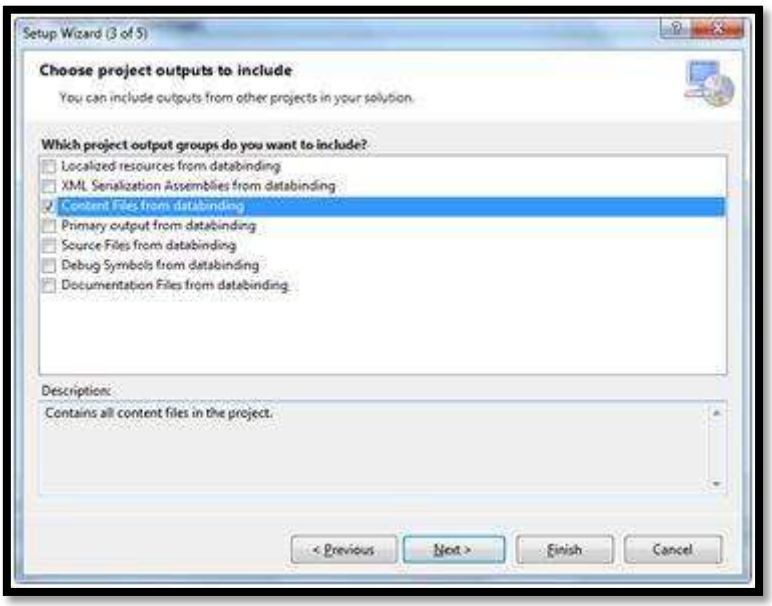
Step (3) : Choosing the default location ensures that the set up project will be located in its own folder under the root directory of the site. Click on okay to get the first splash screen of the wizard.



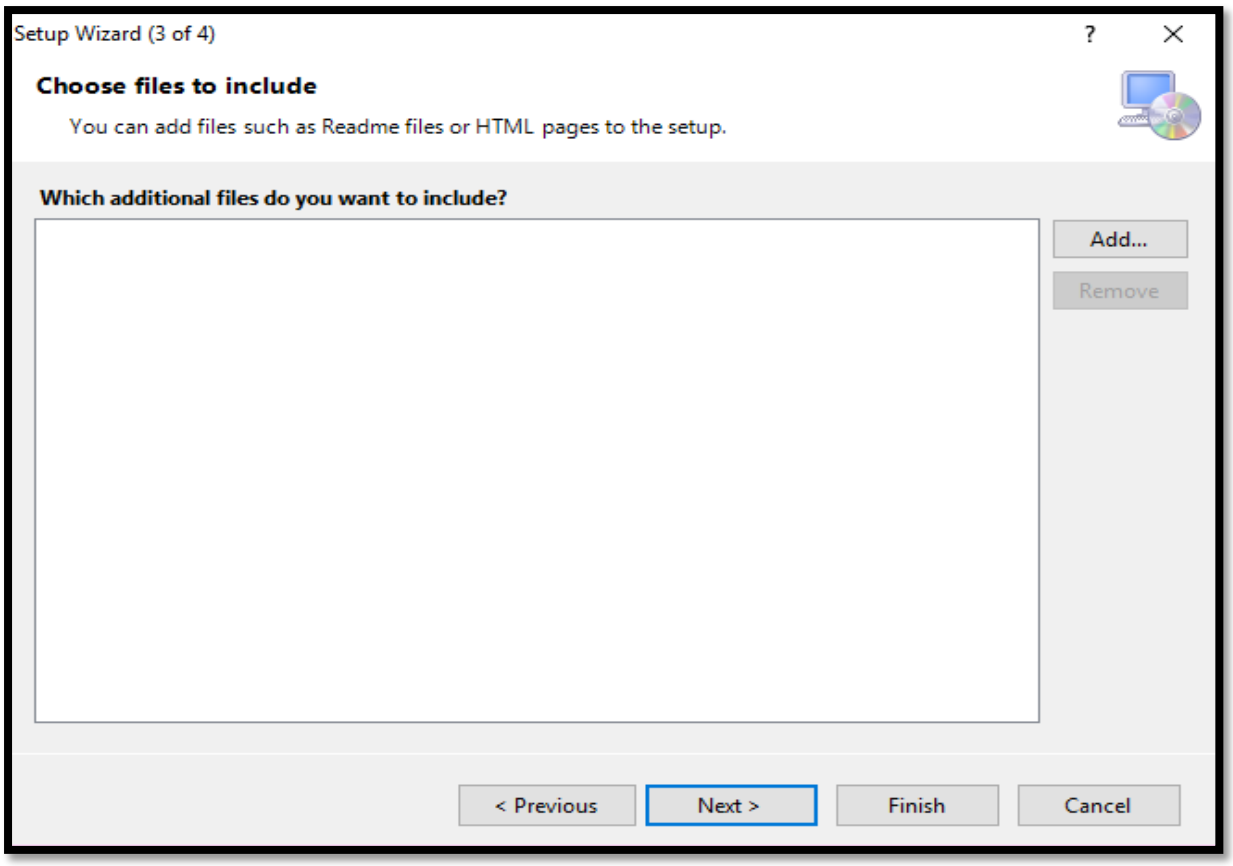
Step (4) : Choose a project type. Select 'Create a setup for a web application'.



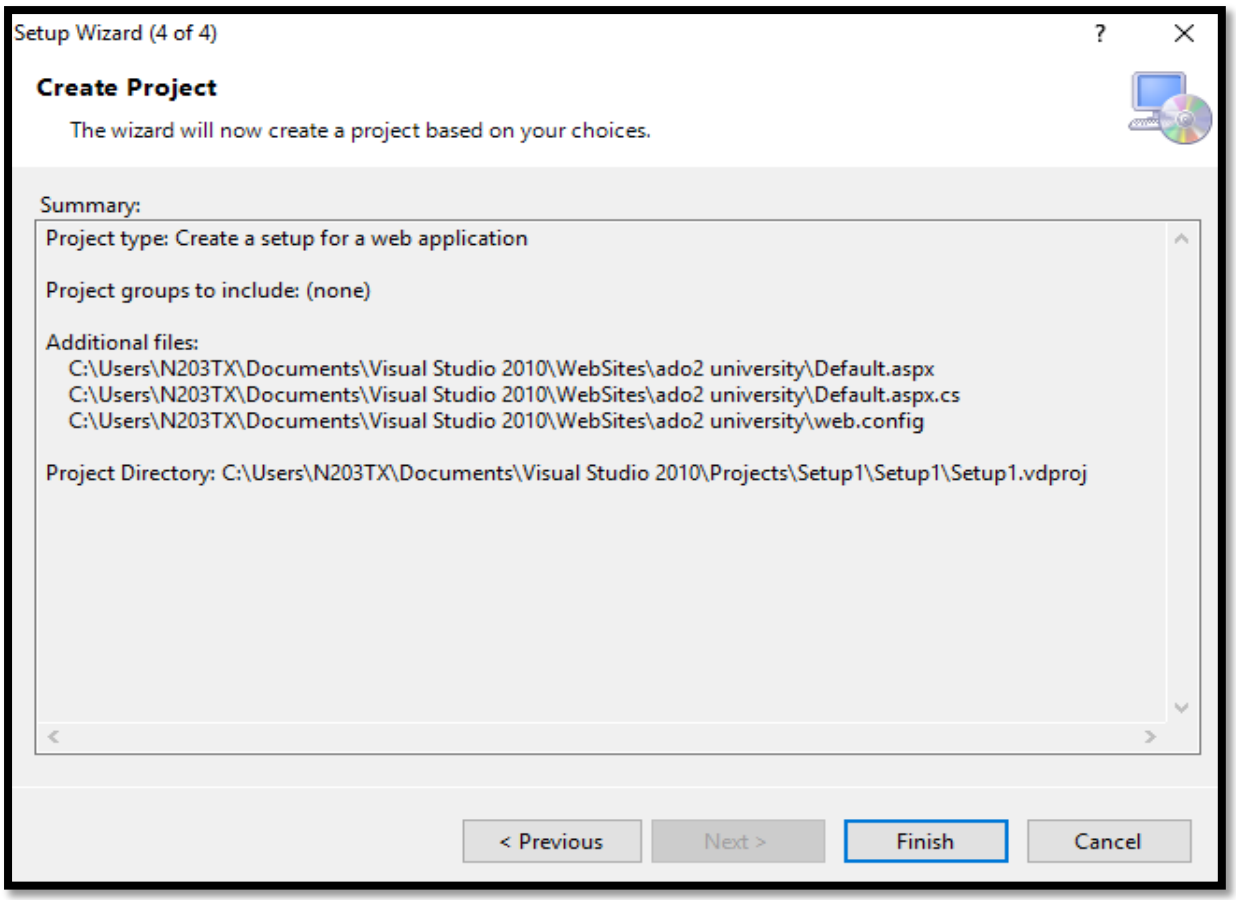
Step (5) : Next, the third screen asks to choose project outputs from all the projects in the solution. Check the check box next to 'Content Files from...'



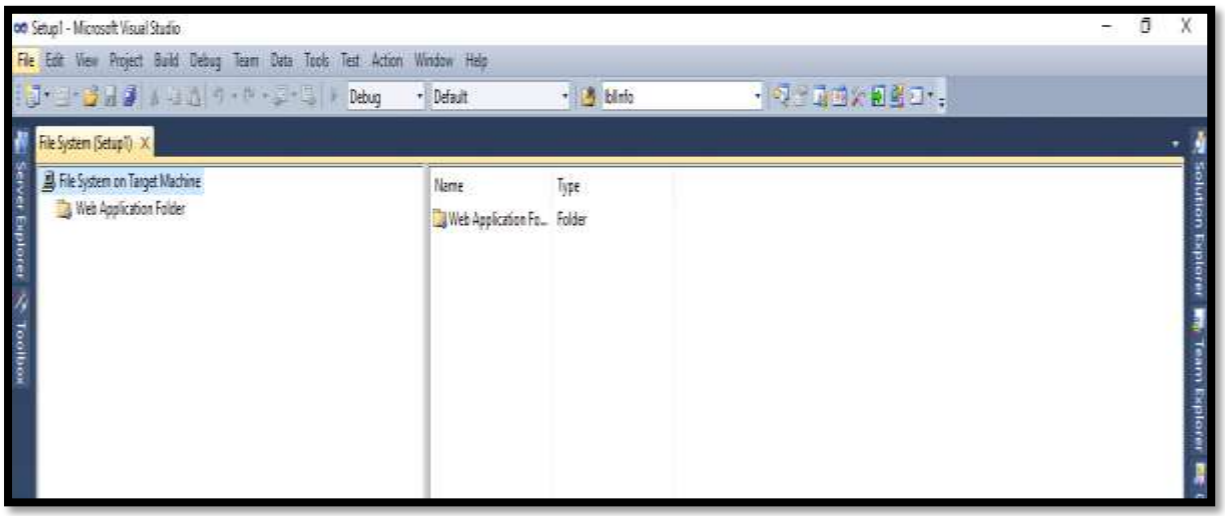
Step (6) : The fourth screen allows including other files like ReadMe. However, in our case there is no such file. Click on finish.



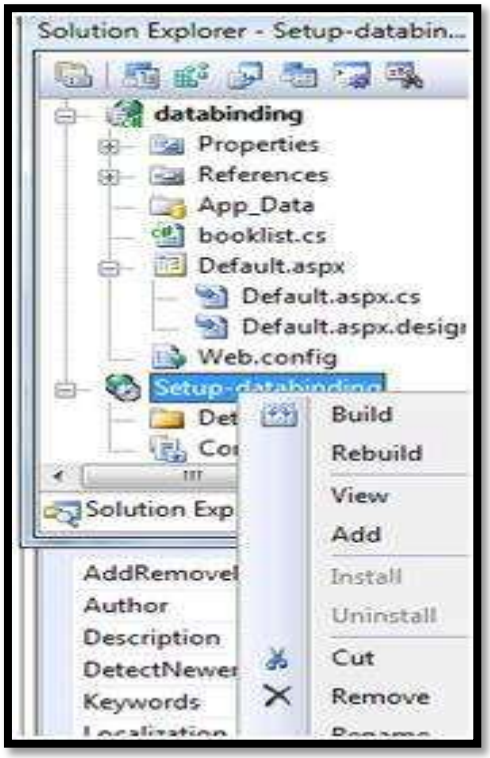
Step (7) : The final screen displays a summary of settings for the set up project.



Step (8) : The Set up project is added to the Solution Explorer and the main design window shows a file system editor.



Step (9) : Next step is to build the setup project. Right click on the project name in the Solution Explorer and select Build.



Step (10) : When build is completed, you get the following message in the Output window:

```
Packaging file 'Web.config'...
Packaging file 'Default.aspx'...
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Two files are created by the build process:

- Setup.exe
- Setup-databinding.msi

You need to copy these files to the server. Double-click the setup file to install the content of the .msi file on the local machine.

12.3 CRYSTAL REPORTS

Crystal Reports is the standard reporting tool for Visual Studio .NET used to display data of presentation quality. You can display multiple-level totals, charts to analyze data, and much more in Crystal Reports. Creating a Crystal Report requires minimal coding since it is created in Designer interface.

Advantages of Crystal Reports

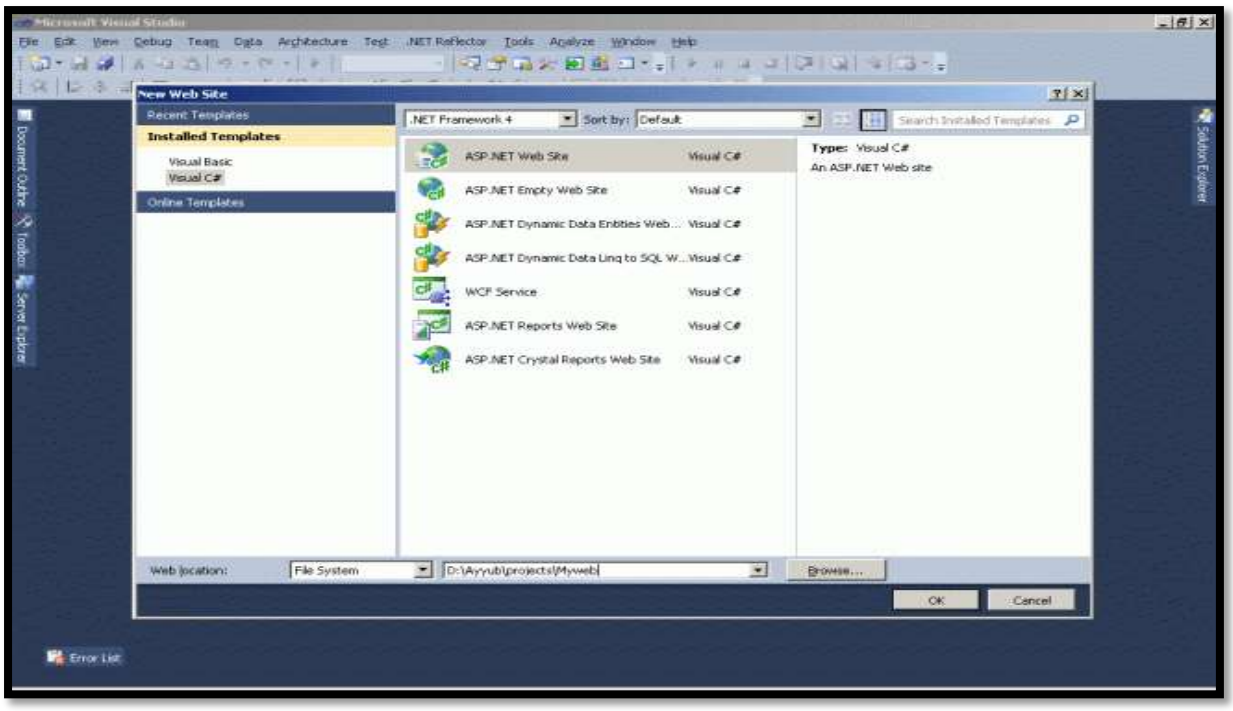
Some of the major advantages of using Crystal Reports are:

1. Rapid report development since the designer interface would ease the coding work for the programmer.
2. Can extend it to complicated reports with interactive charts and enhance the understanding of the business model.
3. Exposes a report object model, can interact with other controls on the ASP.NET Web form.
4. Can programmatically export the reports into widely used formats like .pdf, .doc, .xls, .html and .rtf.
5. Save time using powerful report creation, integration, and delivery tools.

It turns out that Crystal Reports for Visual Studio 2010 will be released separately, instead of included with the product and most importantly, Crystal Reports for Visual Studio 2010 will continue to be free, with no registration required.

Let’s start by creating a new website in Visual Studio 2010.

Open VS 2010, select Visual C# and ASP.NET Web Site and click OK as shown below.

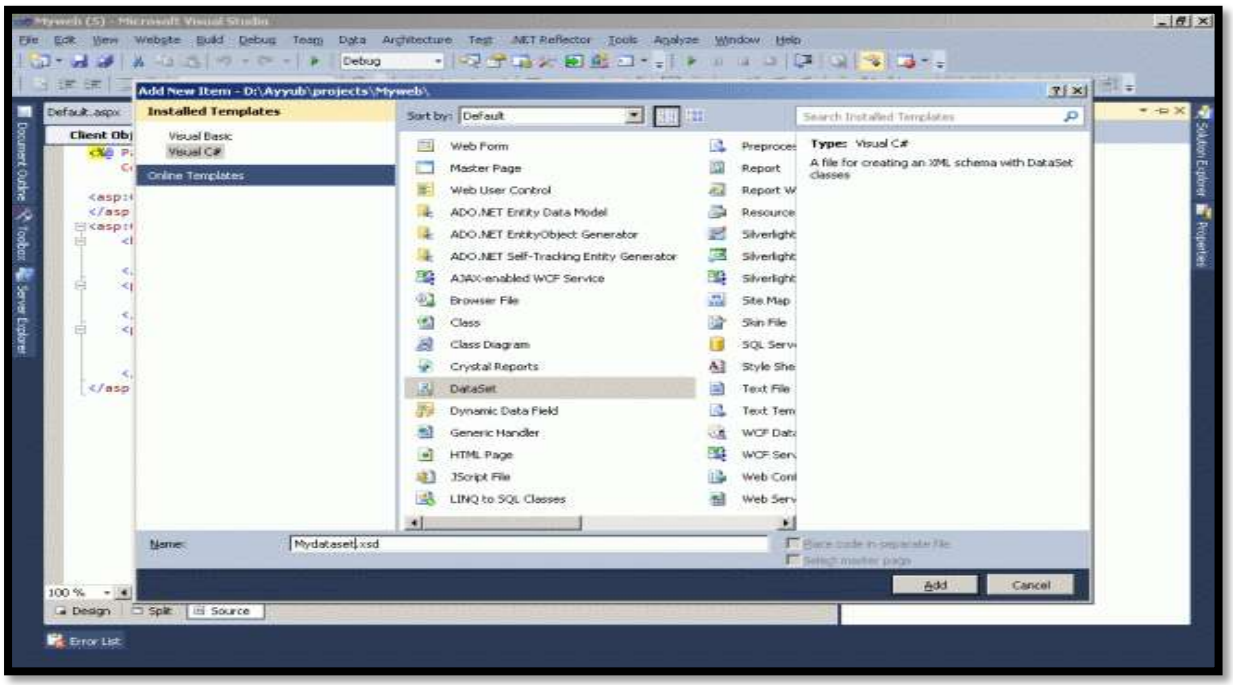


This action will create a new Web site project.

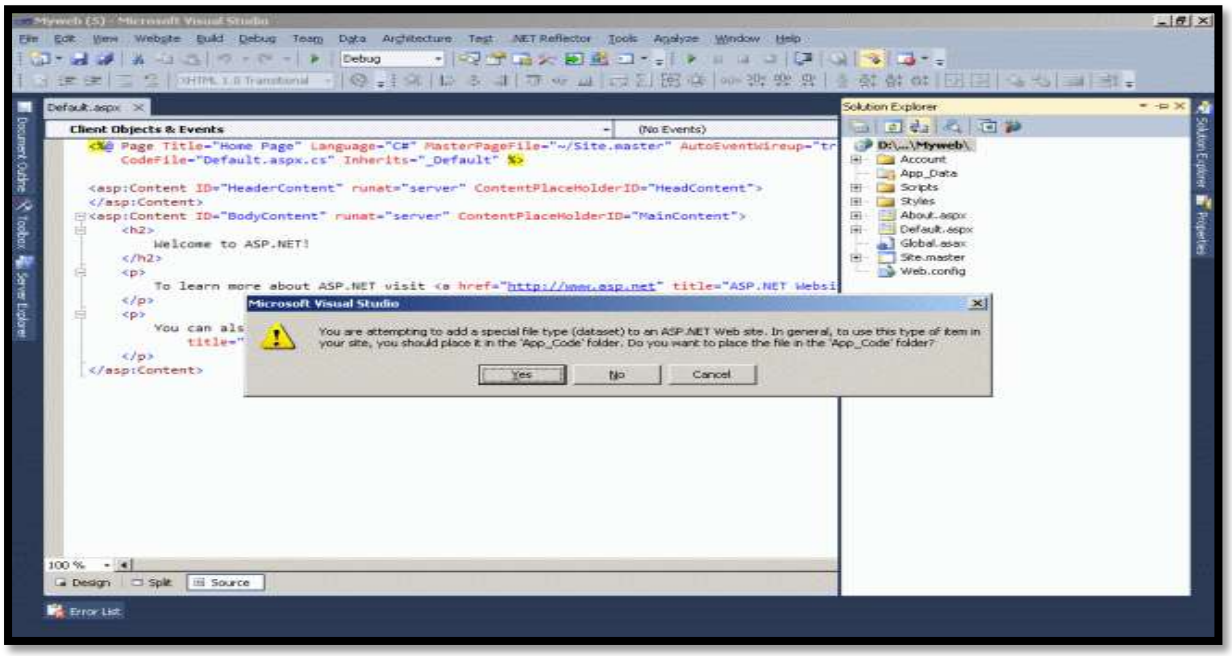
Once we have a Web site project created, next step is to get database access in the project. That we do using a DataSet from a database.

Creation of Dataset (xsd) File

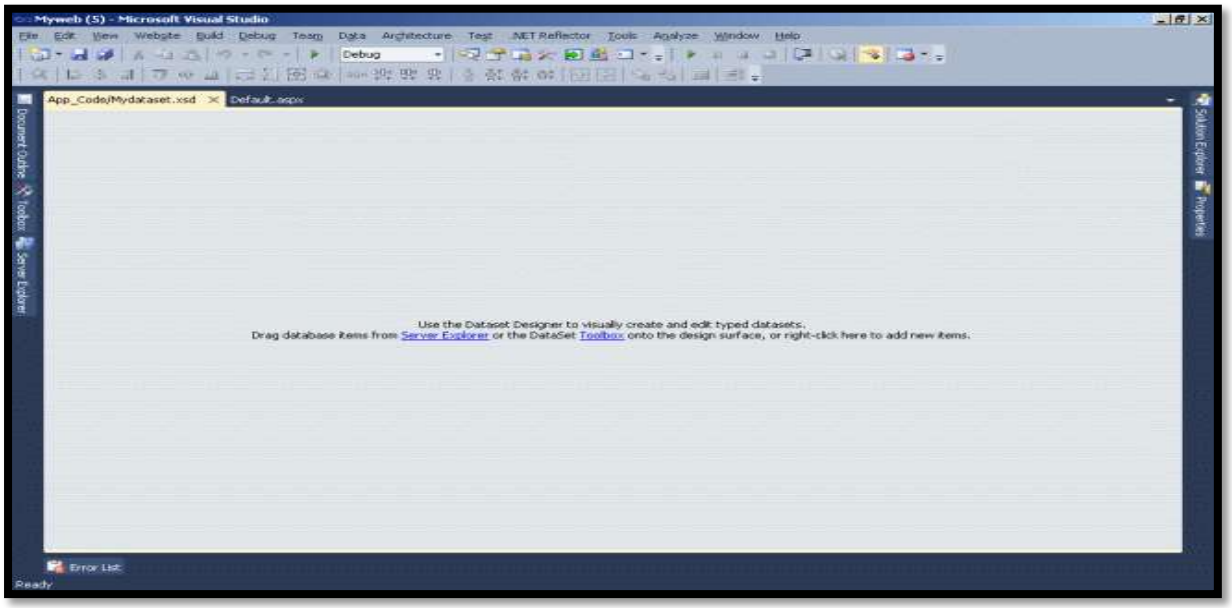
- The following figure shows you the process to create a DataSet file.
- To add a DataSet file, click on Solution Explorer -> Right Click on Project -> click on Add new Item and then it will show you the following screen:



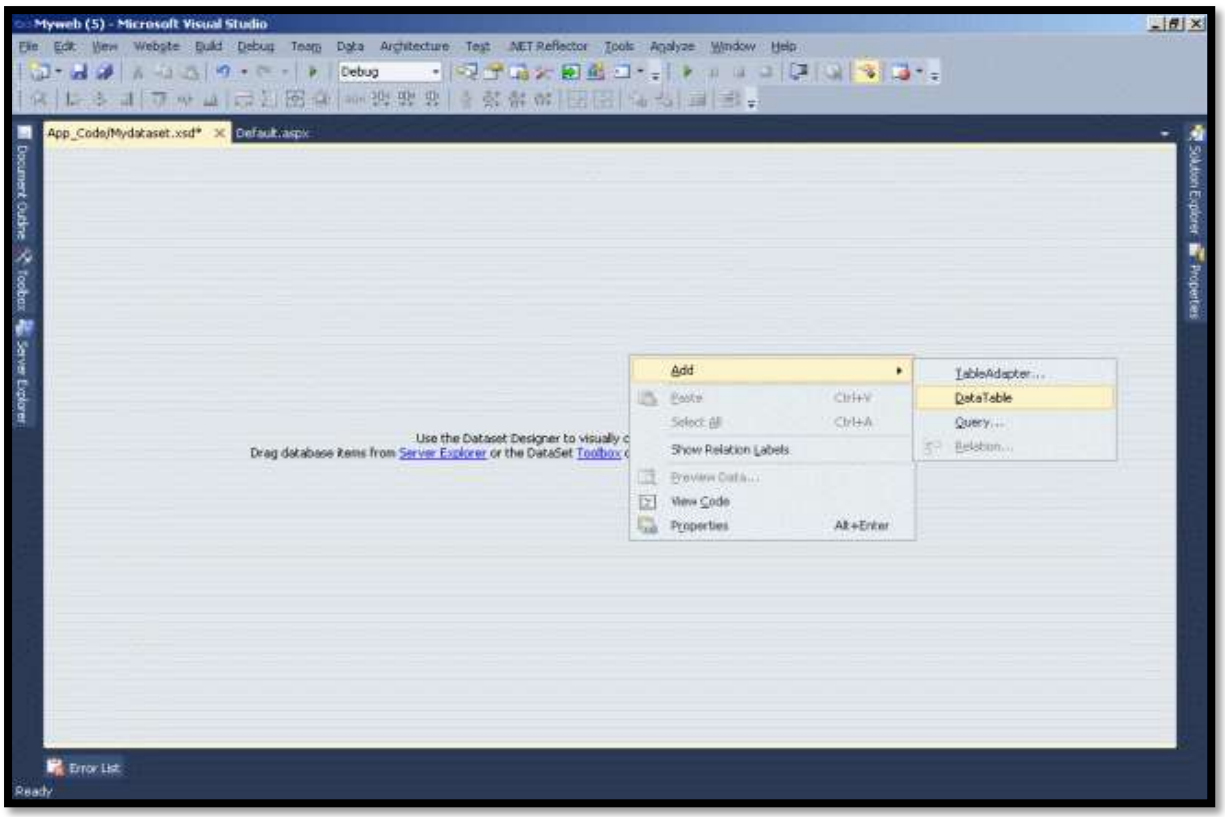
- Enter the Datset file name. Click on the ok button.



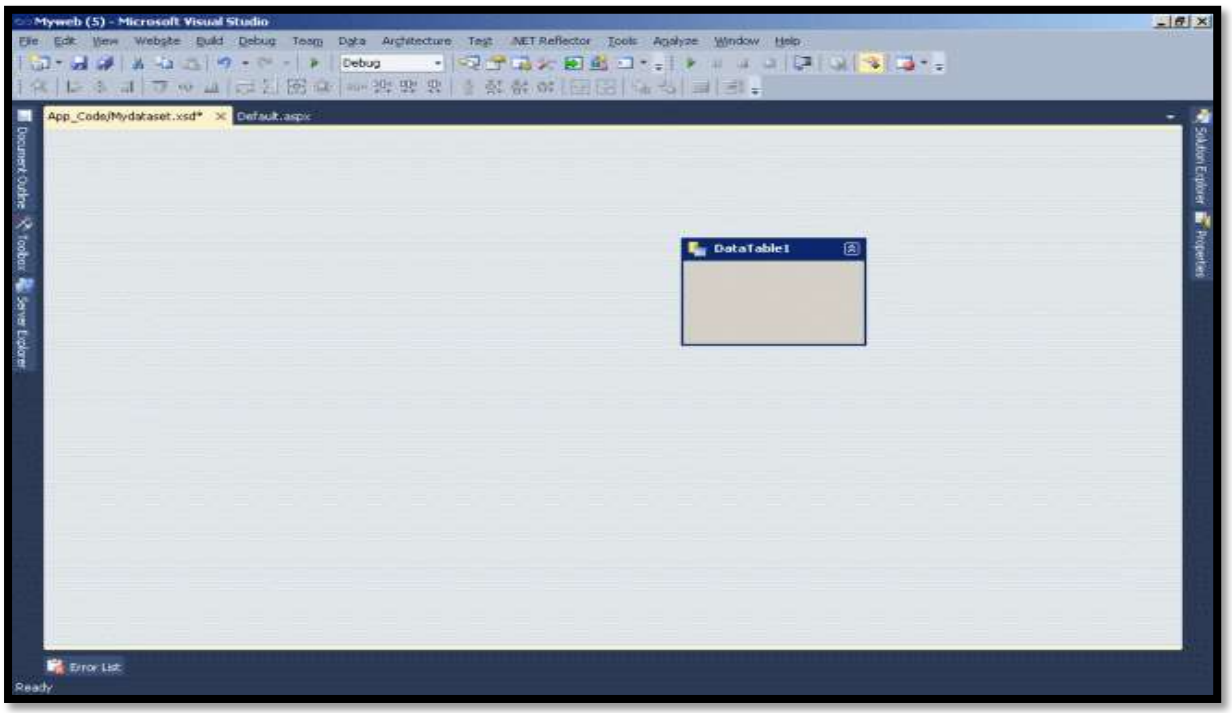
- It will ask for confirmation to put that file in the App_Code folder. Just click yes and that file will opened in the screen as a blank screen.



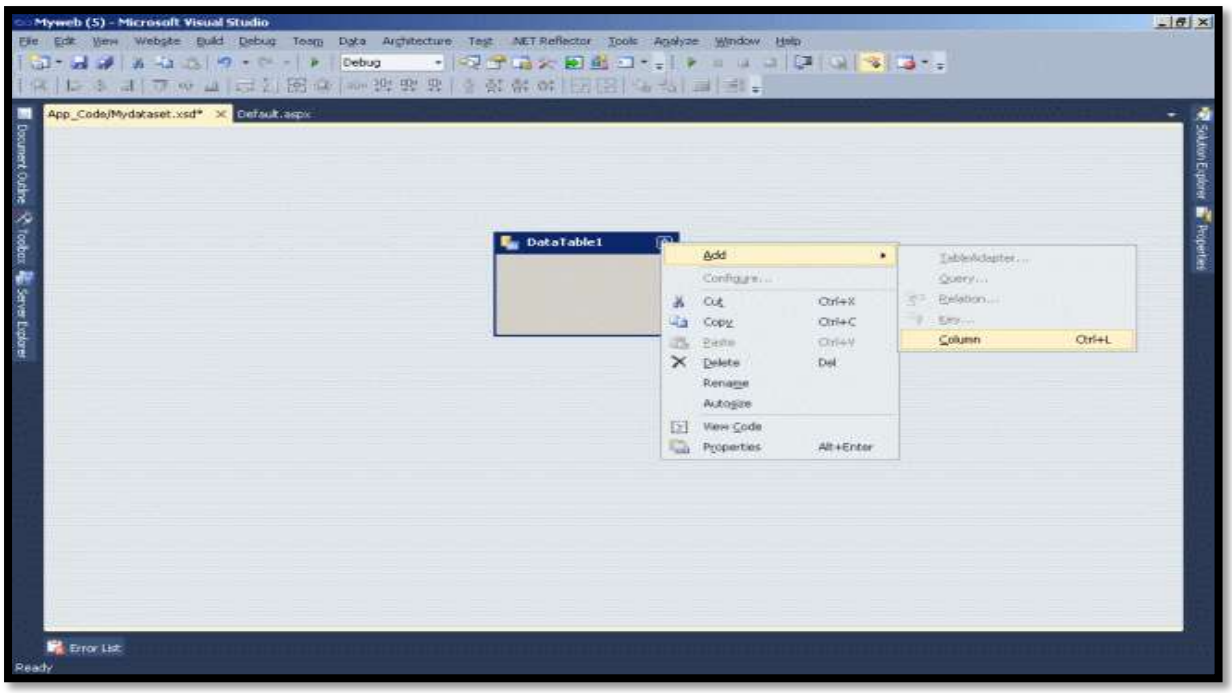
- Now we will add one blank datatable to that mydataset.xsd.
- Right-click in the area of the file and select Add -> Datatable.
- It will add one DataTable1 to the screen.
- The following Figure 5 shows how to add a datatable to the mydataset.XSD file.



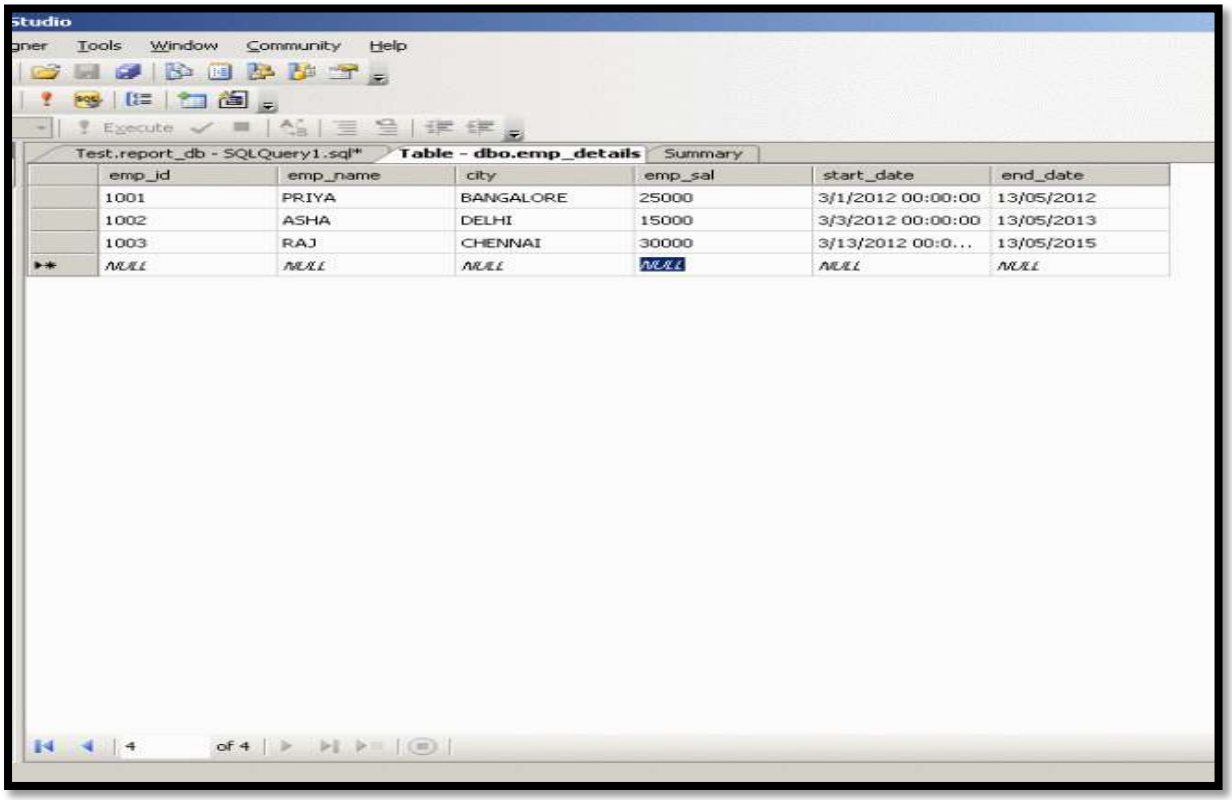
- Now datatable1 is added to XSD file.

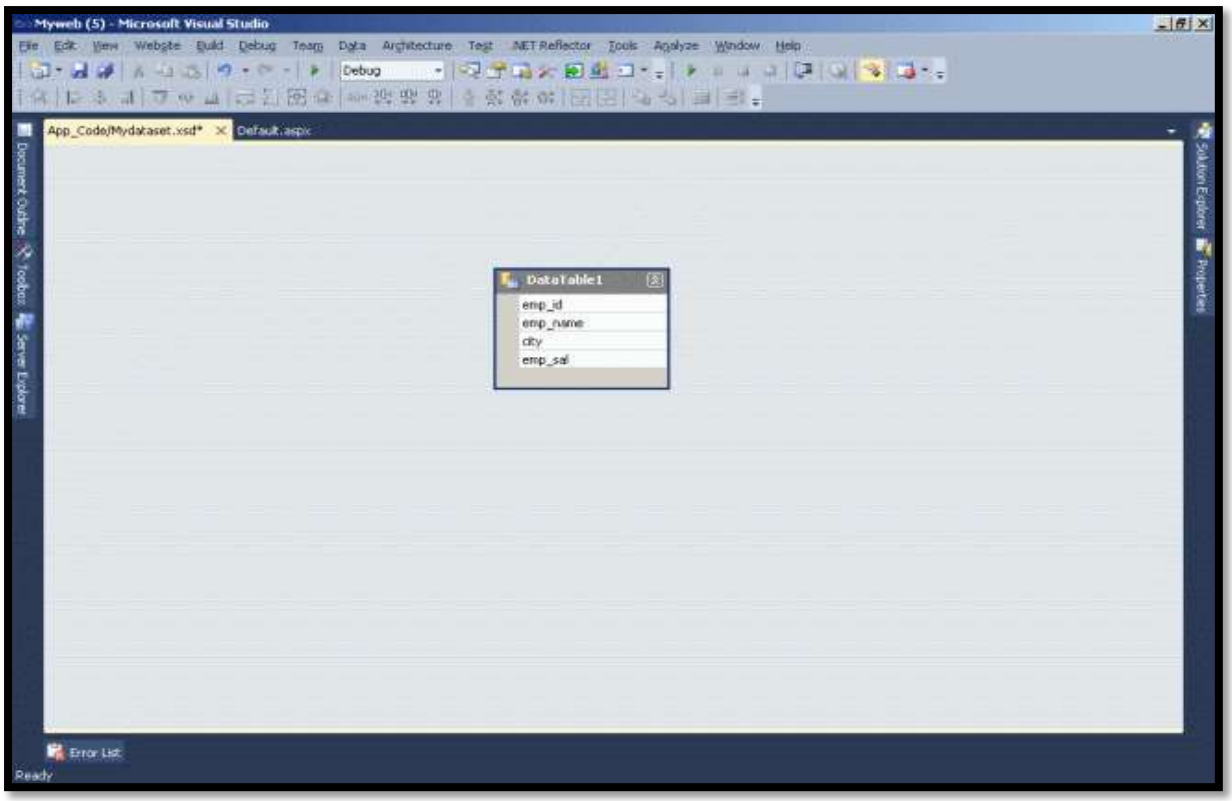


- Now we will add a data column to the datatable1 as per figure 6.
- Remember, whatever columns we add here will be shown on the report.
- So add the columns you want to display in your reports one by one here.

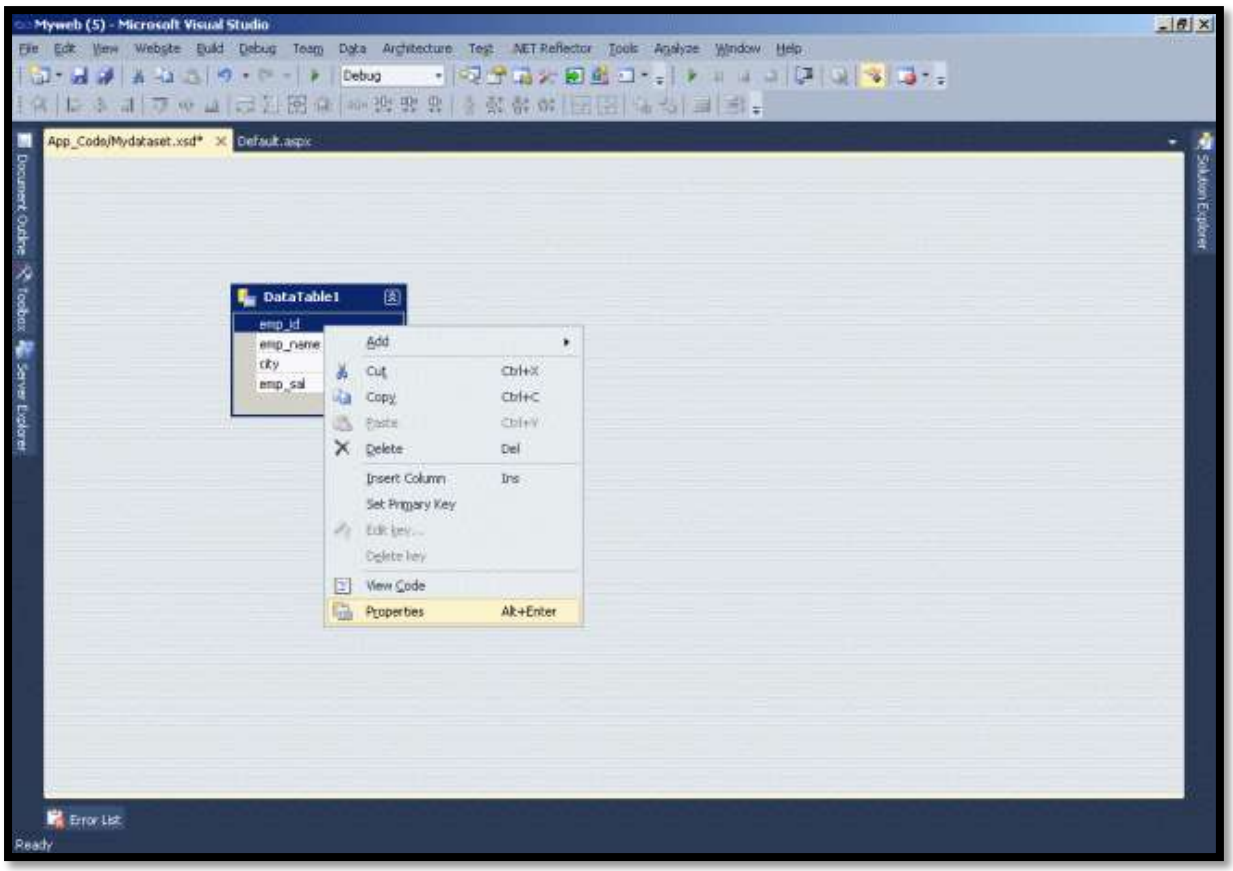


- Always remember to give the same name for the column and data type of column which is the same as the database, otherwise you will get an error for field and data type mismatch.

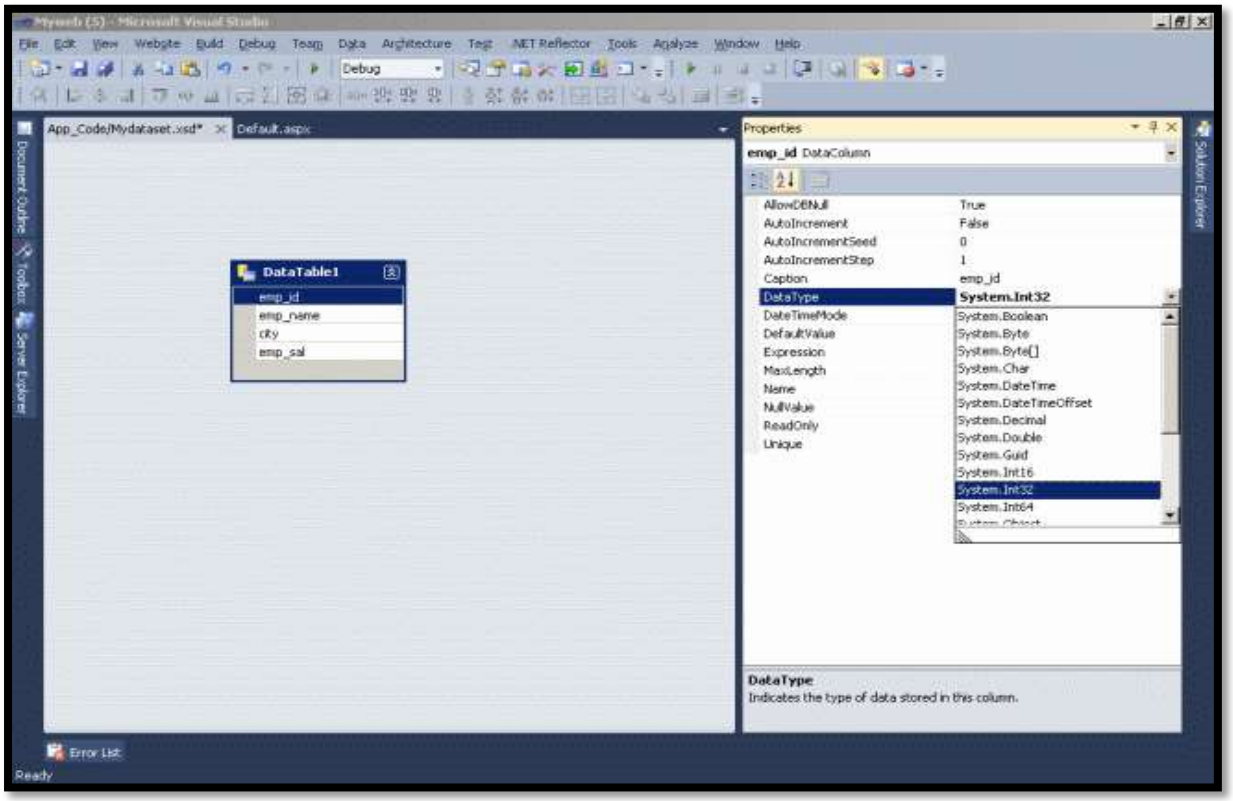




- To set property for the columns the same as the database.
- The following figure will show you how to set the property for the data columns.
- The default data type for all the columns is string.
- To change the data type manually right-click on the datacolumn in the datatable and select property.



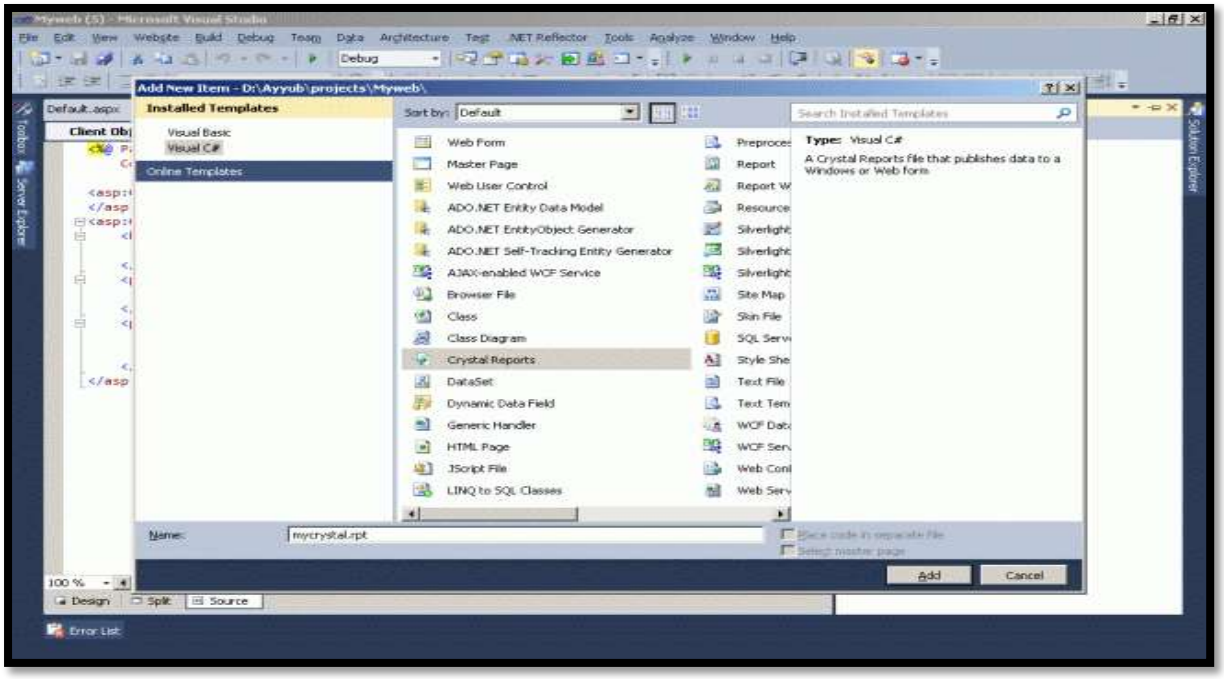
- From the property window, select the appropriate datatype from the DataType Dropdown for the selected datacolumn.



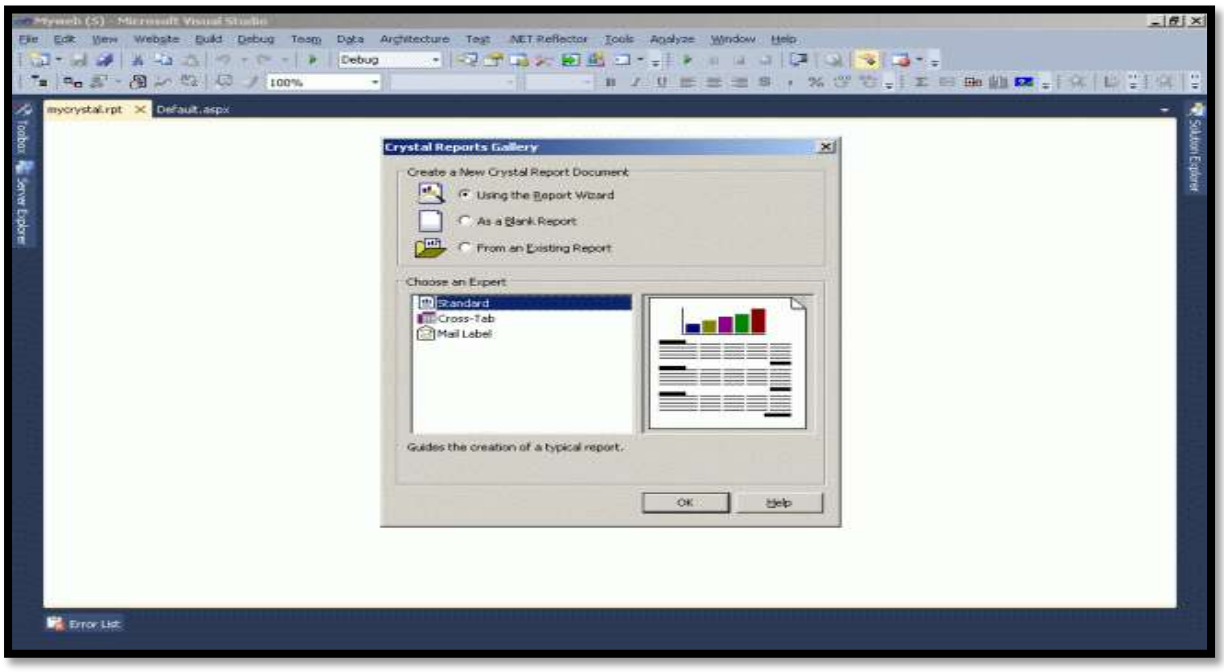
- XSD file creation has been done.
- Now we will move on to create the Crystal Reports design.

Creation of Crystal report design

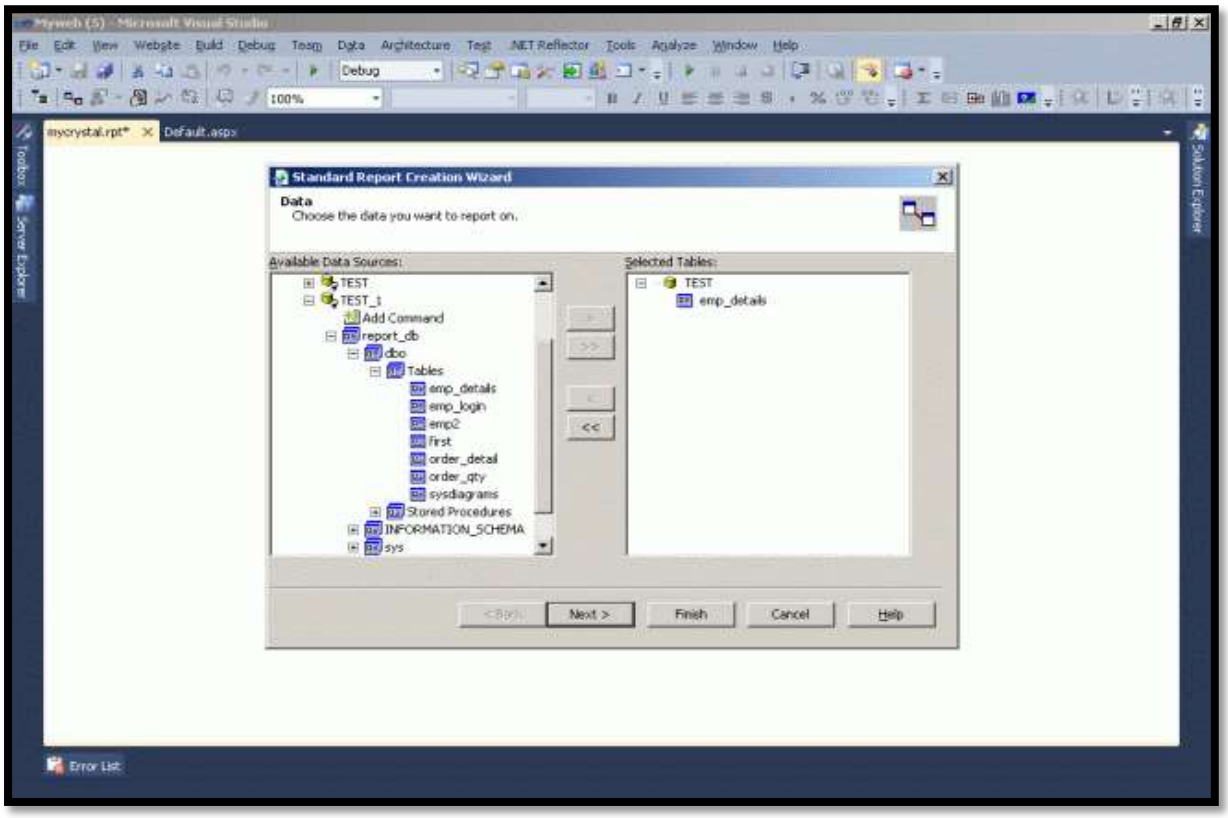
- Click on the Solution Explorer -> Right click on the project name and select Crystal Reports.
- Name it as you choose and click the add button.



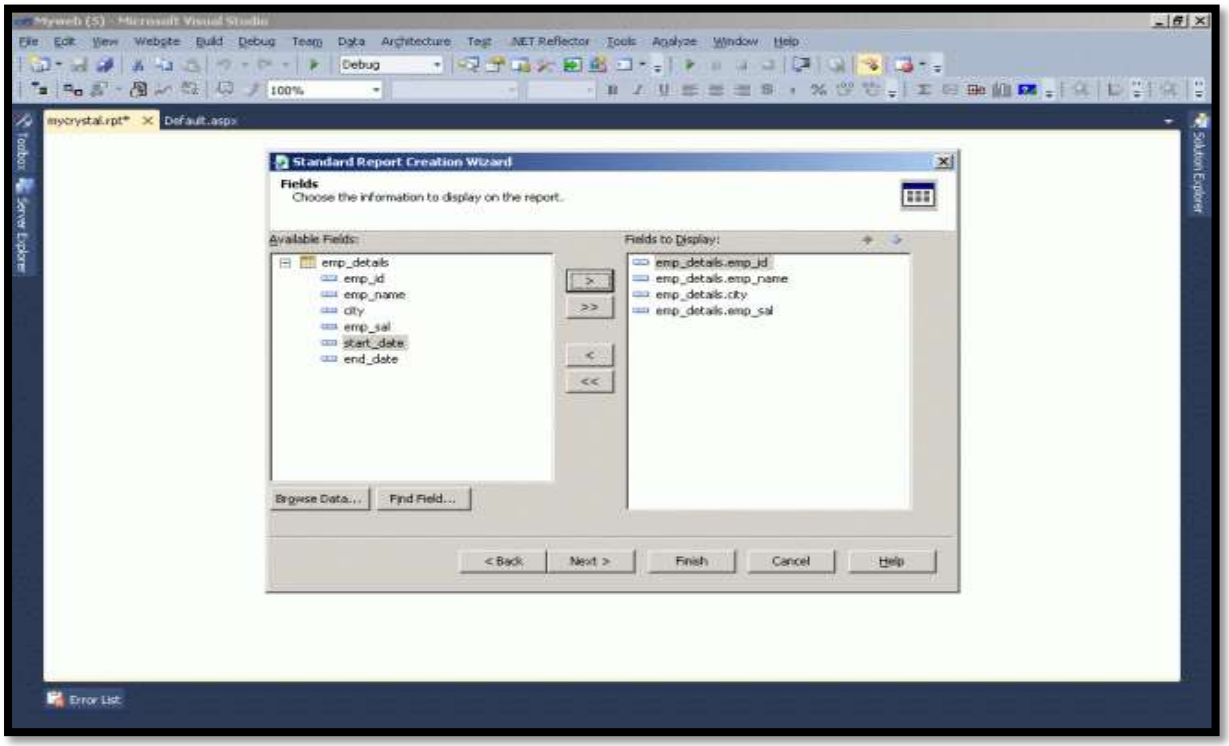
- After clicking on the add button a .rpt file will be added to the solution.
- It will ask for the report creation type of how you want to create the report.



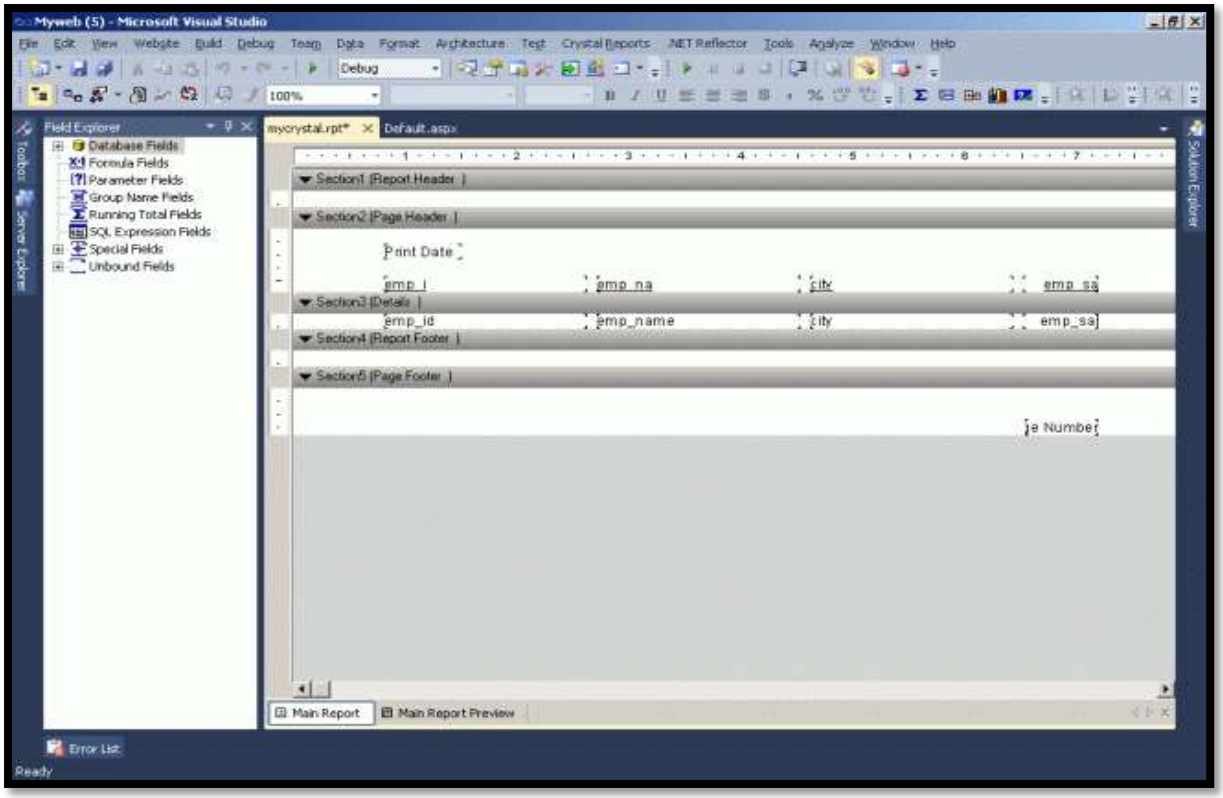
- Click the ok button to proceed.



- Under Data Sources, expand ADO.NET Datasets and select Table and add to the selected table portion located at the right side of the window using the > button. Click on Next.



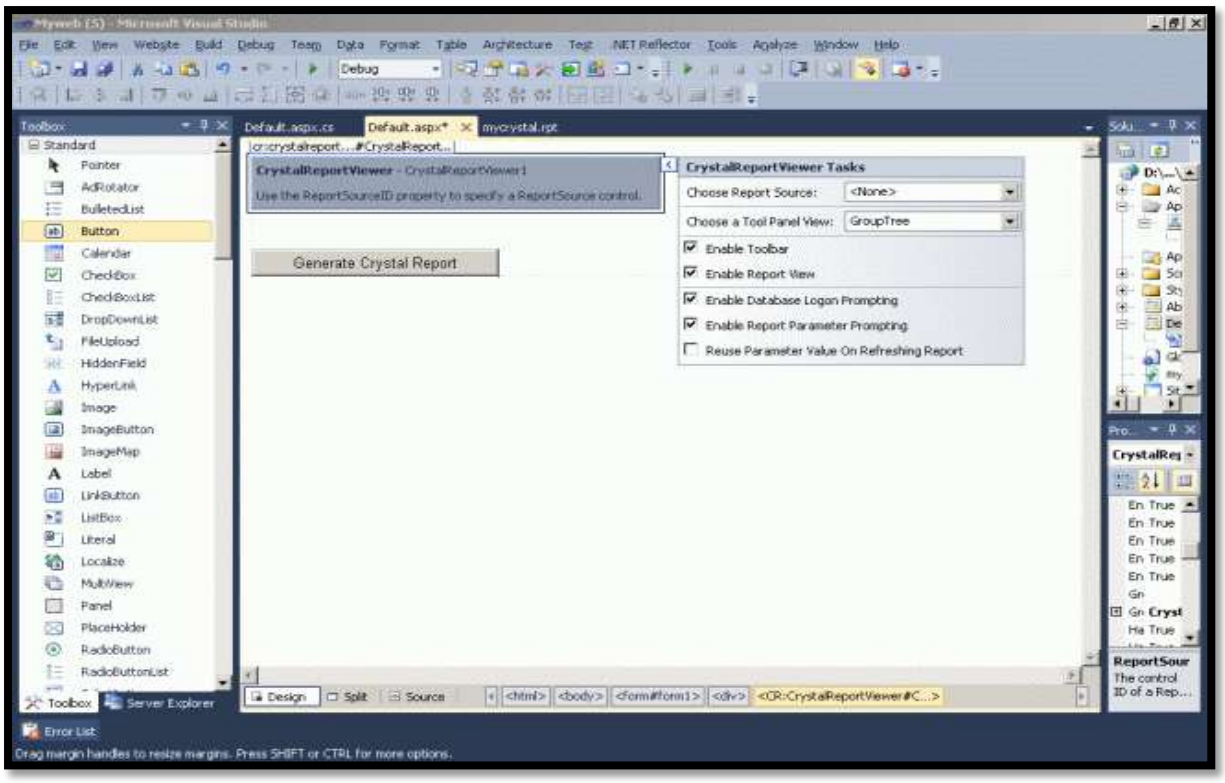
- Select the columns that you want to show in the report.
- Now click on the Finish button and it will show the next screen.



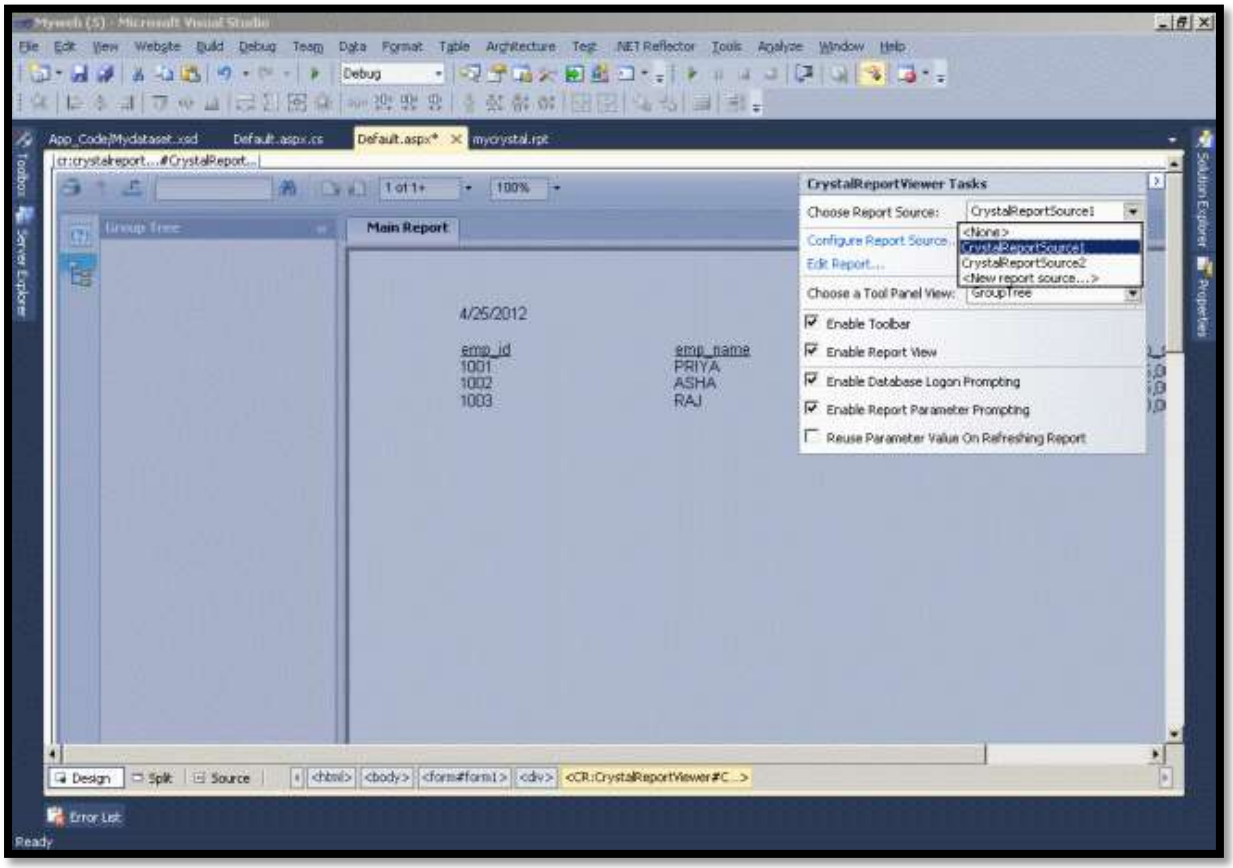
- Once the report file is added, you can see the Field Explorer on the left side of the screen.
- Expand Database Fields, under that you will be able to find the Datatable that we have created earlier.
- Just expand it and drag one by one each field from the Field Explorer to the rpt file the under detail section.
- Now the report design part is over.
- Now we have to fetch the data from the database and bind it to the dataset and then Show that dataset to the report viewer.

Crystal report Viewer

- First Drag a CrystalReportViewer control on the aspx page from the Reporting Section of the tool box.
- Add a command Button.



- Configure the CrystalReportViewer and create a link with Crystal Reports.
- Select the Crystal Reports source from the right side of the control.

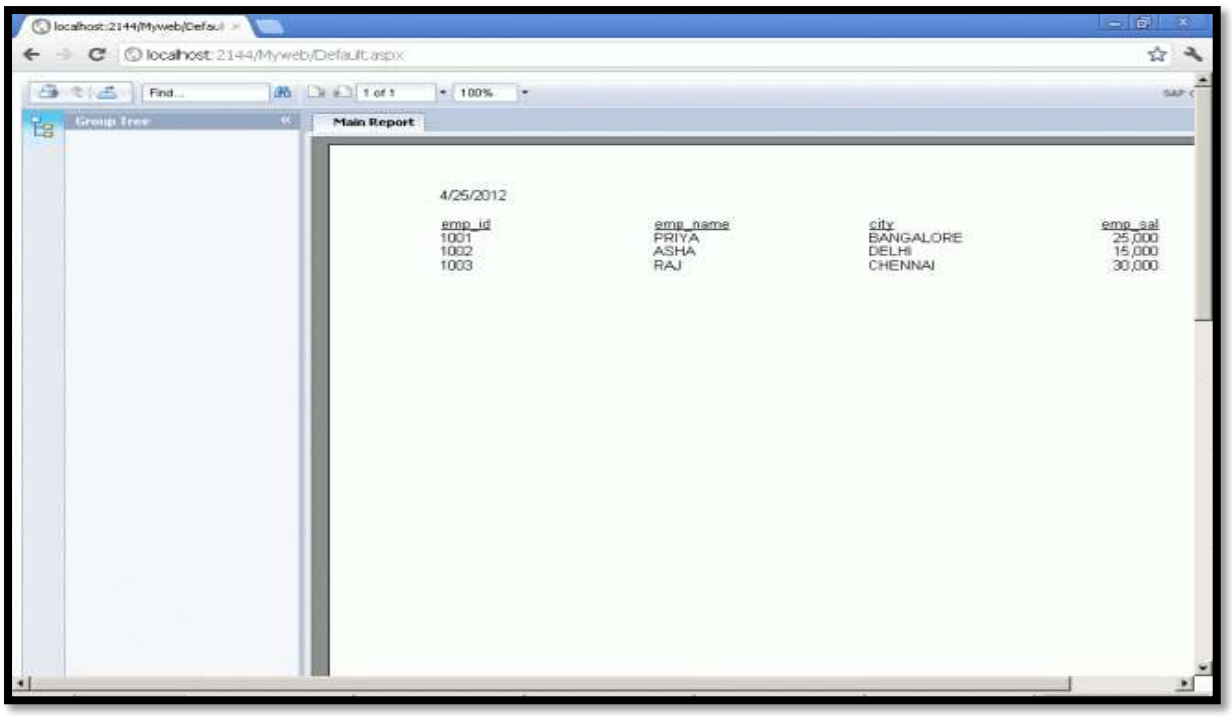


The following is the final code for reports (Default.aspx).

Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using CrystalDecisions.CrystalReports.Engine;
using CrystalDecisions.Shared;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      CrystalReportViewer1.Visible = false;
   }
   protected void cmdcrystal_Click(object sender, EventArgs e)
   {
      CrystalReportViewer1.Visible = true;
      ReportDocument rDoc = new ReportDocument();
      Mydataset dset = new Mydataset(); // dataset file name
      DataTable dtable = new DataTable(); // data table name
      dtable.TableName = "Crystal Report ";  // Crystal Report Name
      rDoc.Load(Server.MapPath("mycrystal.rpt")); // Your .rpt file path
      rDoc.SetDataSource(dset); //set dataset to the report viewer.
      CrystalReportViewer1.ReportSource = rDoc;
   }
}
```

OutPut



12.4 SUMMARY

This unit gives an overview of DATA CONTROL and DATA SOURCE in ADO.NET such as GridView, FormView and DetailsView etc. Further it discusses about deployment of ASP.NET web application and crystal reports.

12.5 EXERCISE

1. What is data source control? Explain various data source control in .net.
2. What is GridView Control? Explain operation of GridView.
3. Explain FormView control with example.
4. Explain DetailsView control with example.
5. Briefly explain FormView control. How is it different from DetailsView?
6. Explain the deployment of asp.net website with its steps.
7. What is crystal report? How we can create a crystal report in .net.

REFERENCE

1. ADO.NET: The Complete Reference
2. Beginning ASP.NET 4 by Imar Spaanjaars
3. Database Programming with Visual Basic .Net and ADO.NET by F Scott Barker, Publisher: Pearson Education
4. https://www.c-sharpcorner.com/
5. https://stackoverflow.com/
6. https://www.tutorialspoint.com
7. https://www.sap.com/india/products/crystal-visual-studio.html

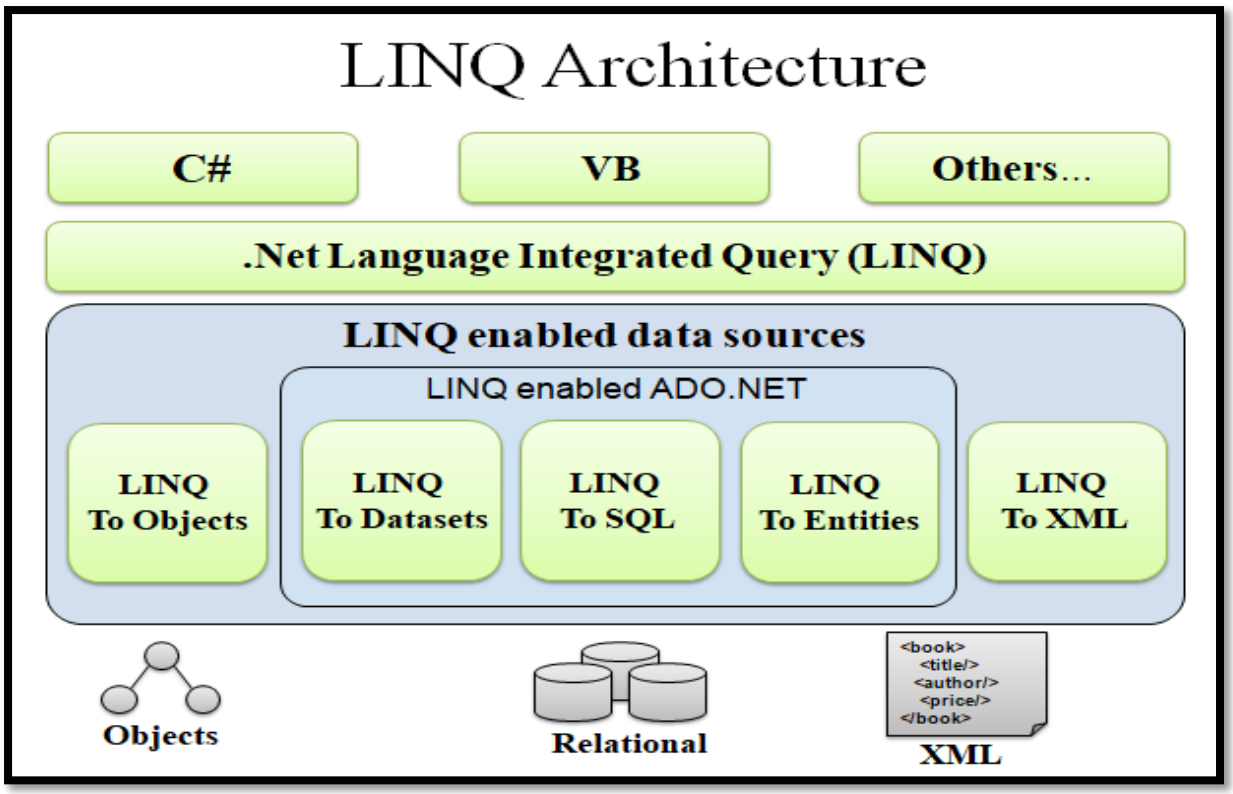
LINQ

Content

13.0 LINQ - LANGUAGE INTEGRATED QUERY

LINQ stands for Language Integrated Query. LINQ is a data querying methodology which provides querying capabilities to .NET languages with syntax similar to a SQL query. LINQ is a uniform programming model for querying and manipulating data with a consistent model from any data source. LINQ is just another tool for embedding SQL queries into code. Extends powerful query capabilities to C#, VB.NET languages.In a LINQ query, you are always working with objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, ADO.NET Datasets, .NET collections and any other format for which a LINQ provider is available.

LINQ simplifies the working model with its generic architecture to support many kinds of data sources and provide a common platform to execute the query and get the results.



A .Net application uses LINQ queries to communicate with various kinds of data sources like SQL Server, XML documents and in-memory objects. Between SQL queries and data sources one layer of LINQ providers are present that converts the LINQ queries into the format that the underlying data source can understand.

13.1 LINQ FEATURES

- Language Integration
- Single general purpose Declarative Programming
- Standard Query Operators that allow
  - traversal, filter, and projection operations
- Transparency Across Different Type Systems
  - Query expressions benefit from
    1. Rich Metadata
    2. Compile-Time syntax checking
    3. Static Typing
    4. IntelliSense

13.1.1Advantages and disadvantages of LINQ

Advantages of LINQ

- LINQ can be used for querying multiple data sources such as relational data and XML data.
- LINQ is a technique for querying data across various kinds of data sources and formats.

- LINQ has syntax highlighting and IntelliSense that help to identify compile-time error checking.
- LINQ is extensible so new types of data sources can be made querable.
- It reduces the complexity of the code and makes it much easy for the program to read.
- LINQ is composable in nature and it can be used to solve complex problems into a series of short, comprehensible queries that are easy to debug.
- The .NET application can interact with LINQ for database operations and LINQ takes care of complete database operations.
- LINQ is declarative, it is very easy to understand and maintain.
- One of the most advantages in using LINQ is that its availability over any .NET platform language such as C#.net, VB.NET and F#.NET.

Disadvantages of LINQ

- LINQ architecture has another layer for LINQ providers that will provide performance overhead sometimes with complex queries.
- LINQ is not a precompiled statement like Stored Procedures.
- Frequent changes must be recompiled and have a deployment overhead for the entire code.
- No good way to view permissions.
- When database queries are converted from SQL to the application side, joins are very slow that are very specific to LINQ to SQL.

13.1.2 Difference between SQL and LINQ

SQL	LINQ
It is stand for STRUCTURE QUERY LANGUAGE.	It is stand for LANGUAGE INTREGRATED QUERY.
SQL Queries: for single inline queries.	LINQ Queries: Linq are difficult to debug but easy to write.
SQL is in most cases a significantly less productive querying language.	LINQ is in most cases a significantly more productive querying language.
SQL is difficult to understand.	Compared to SQL, LINQ is simpler, tidier, and higher-level.
SQL enabling you to access and query from a RDBMS.	LINQ enabling you to access and query a wide variety of sources including collections in your own code, XML files, .NET Datasets, and databases from your VB.NET or C# code.
SQL language is used to create, modify and retrieve information from RDBMS.	LINQ is a uniform programming model for any kind of data access.

The LINQ providers those are included with .NET 4:

- **LINQ to Objects:** This is the simplest form of LINQ. It allows you to query collections of in-memory objects (such as an array, an ArrayList, a List, a Dictionary, and so on).

- **Parallel LINQ:** This is a variation of LINQ to objects that has built-in support for multithreaded execution.
- **LINQ to DataSet:** This form of LINQ resembles LINQ to objects, except it digs DataRow objects out of a DataTable.
- **LINQ to XML:** This form of LINQ allows you to search the elements contained in an XElement or XDocument.
- **LINQ to SQL:** This is the original LINQ provider for data access. It allows you to fetch data from a SQL Server database.
- **LINQ to Entities:** Like LINQ to SQL, LINQ to Entities allows you to perform database queries with a LINQ expression.

13.2 LINQ QUERY OPERATORS

Standard Query Operators in LINQ are actually extension methods for the IEnumerable<T> and IQueryable<T> types. They are defined in the System.Linq.Enumerable and System.Linq.Queryable classes. There are over 50 standard query operators available in LINQ that provide different functionalities like filtering, sorting, grouping, aggregation, concatenation, etc.

Project	**Select** <*expr*>
Filter	**Where** <*expr*>, **Distinct**
Test	**Any(**<*expr*>**)**, **All(**<*expr*>**)**
Join	<*expr*> **Join** <*expr*> **On** <*expr*> **Equals** <*expr*>
Group	**Group By** <*expr*>, <*expr*> **Into** <*expr*>, <*expr*> **Group Join** <*decl*> **On** <*expr*> **Equals** <*expr*> **Into** <*expr*>
Aggregate	**Count(**<*expr*>**)**, **Sum(**<*expr*>**)**, **Min(**<*expr*>**)**, **Max(**<*expr*>**)**, **Avg(**<*expr*>**)**
Partition	**Skip** [**While**] <*expr*>, **Take** [**While**] <*expr*>
Set	**Union**, **Intersect**, **Except**
Order	**Order By** <*expr*>, <*expr*> [**Ascending** **|** **Descending**]

13.3 LINQ SYNTAX

There are two syntaxes of LINQ. These are the following ones.

Lamda (Method) Syntax

var longWords = words.Where(w ⇒ w.length > 10);

Query (Comprehension) Syntax

var longwords = from w in words where w.length > 10;

from [identifier] in [source collection]
let [expression]
where [Boolean expression]
order by [[expression](ascending/descending)], [optionally repeat]
select [expression]
group [expression] by [expression] into [expression]

Where,
from / in - Specifies the data source
where - Conditional Boolean expression
order by (ascending/descending) - Sorts the results into ascending or descending order
select - Adds the result to the return type
group / by - Groups the results based on a given key

13.4 LINQ QUERY EXPRESSION

- A Query is a set of instructions that describe what the data is to retrieve from a given data source.
- A Query Expression is a query, expressed in query syntax.
- A LINQ Query Expression is also very similar to SQL.
- A LINQ Query Expression contains the following three (3) clauses:
  1. ***From***
  2. ***Where***
  3. ***Select***
- **From:** specifies the data source.
- **Where:** applies data filtration.
- **Select:** specifies the returned elements.

13.5 TYPES OF LINQ

The types of LINQ are mentioned below in brief.

- LINQ to Objects
- LINQ to XML(XLINQ)
- LINQ to DataSet
- LINQ to SQL (DLINQ)
- LINQ to Entities

13.5.1 LINQ to Objects

- It is the use of LINQ queries with any IEnumerable or IEnumerable(T) collection directly, without the use of an intermediate.
- LINQ provider or API such as LINQ to SQL or LINQ to XML.
- It allows query any enumerable collections such as
  - List(T),
  - Array, or
  - Dictionary(TKey, TValue).
- There are also many advantages of LINQ to Objects over traditional foreach loops like more readability, powerful filtering, capability of grouping, enhanced ordering with minimal application coding. Such LINQ queries are also more compact in nature and are portable to any other data sources without any modification or with just a little modification.

Example

This example includes LINQ operator, LINQ Expression, LINQ filter and LINQ sorting.

```
using System;
using System.Linq;
namespace LINQTOOBJECT
{
   class Program
   {
      static void Main(string[] args)
      {
         // LINQ TO OBJECT
string[] names = { "ZAIDI", "MUZAFFAR", "SAMEER", "ARIF","ZEHRA",
"MOHADDESA" , "RIZVI" };
         var displayname = from name in names
                           where name.Contains("A")
                          //where name.StartsWith("Z") // filters
                          //where name.EndsWith("R")
                          orderby name  // sorting
                          select name;

         foreach (string sname in displayname)
            {
```

```
            Console.WriteLine("NAME : {0} ", sname);
        }
        Console.ReadLine();

    }
  }
}
```

Output:
NAME : ARIF
NAME : MOHADDESA
NAME : MUZAFFAR
NAME : SAMEER
NAME : ZAIDI
NAME : ZEHRA

13.5.2 LINQ to XML:

- LINQ to XML provides an in-memory XML programming interface that leverages the .NET Language-Integrated Query (LINQ) Framework.
- LINQ to XML is a LINQ-enabled, in-memory XML programming interface that enables you to work with XML from within the .NET Framework programming languages.

There are many more classes that can be used in LINQ to XML. The following are a few of them to explain the functional construction of XML.

- XDocument
- XDeclaration
- XComment
- XElement
- XAttribute

Example

EMPLOYEE.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Employees>
  <Employee>
    <FirstName>ZAIDI</FirstName>
    <Age>29</Age>
    <Dept>Computer Science</Dept>
    </Employee>
  <Employee>
    <FirstName>SAIF</FirstName>
   <Age>30</Age>
   <Dept>Information Technology</Dept>
```

```
  </Employee>
  <Employee>
    <FirstName>ARIF</FirstName>
    <Age>48</Age>
    <Dept>Engineering</Dept>
    </Employee>
  <Employee>
    <FirstName>SOHRABH</FirstName>
    <Age>30</Age>
    <Dept>M.Sc - IT</Dept>
    </Employee>
</Employees>
```

Default.aspx Code:

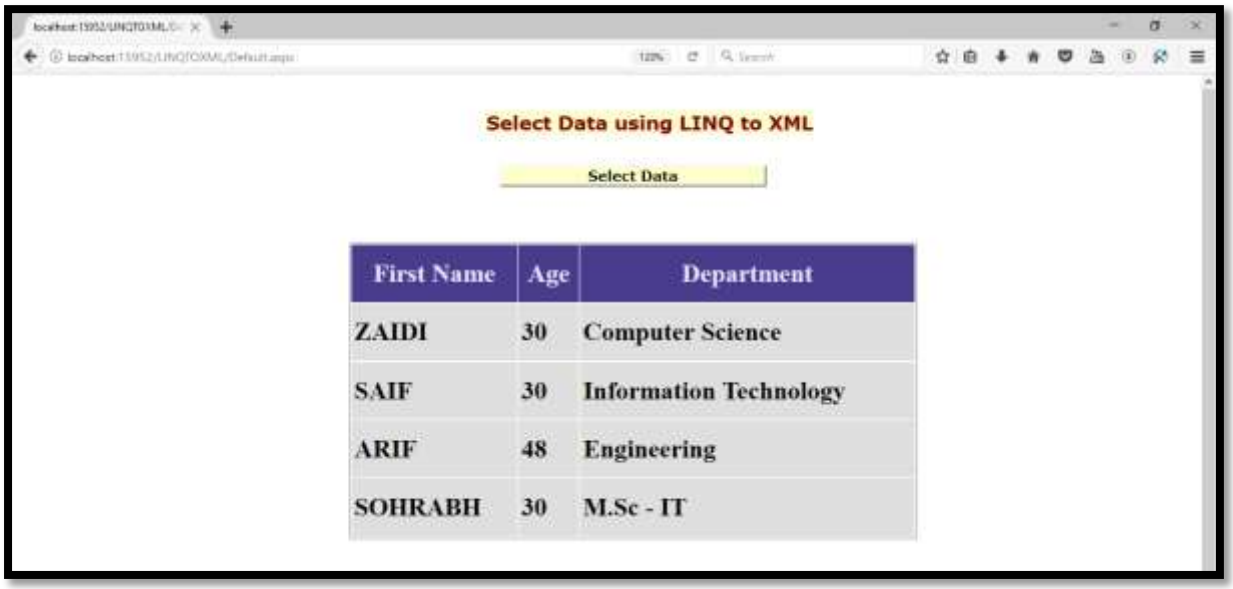
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table class="style1">
            <td>
                <asp:Label ID="Label1" runat="server" Text="Select Data using LINQ to
XML" Font-Bold="True" Font-Size="Large" Font-Names="Verdana" ForeColor="Maroon"
BackColor="#66FFFF"></asp:Label></td>
            <td>
                <asp:Button ID="Button1" runat="server" Text="Select Data" Font-
Names="Verdana" Width="253px" BackColor="Lime" Font-Bold="True"
OnClick="Button1_Click" /></td>
            <td>
                <asp:GridView ID="GridView1" runat="server" BackColor="White"
BorderColor="#336666" BorderWidth="3px" CellPadding="4"
GridLines="Horizontal" AutoGenerateColumns="False"
BorderStyle="Double" Height="186px" Width="260px"
onselectedindexchanged="GridView1_SelectedIndexChanged">
<FooterStyle BackColor="White" ForeColor="#333333"></FooterStyle>

<HeaderStyle BackColor="#336666" Font-Bold="True"
ForeColor="White"></HeaderStyle>
<PagerStyle HorizontalAlign="Center" BackColor="#336666" ForeColor="White">
</PagerStyle>
<RowStyle BackColor="White" ForeColor="#333333" />
<SelectedRowStyle BackColor="#339966" ForeColor="White" Font-
Bold="True"></SelectedRowStyle>
<Columns>
```

```
<asp:BoundField DataField="FirstName" HeaderText="First Name" ReadOnly="true" />

<asp:BoundField DataField="Age" HeaderText="Age" ReadOnly="true" />

<asp:BoundField DataField="Dept" HeaderText="Department" ReadOnly="true" />
</Columns>
<SortedAscendingCellStyle BackColor="#F7F7F7" />
<SortedAscendingHeaderStyle BackColor="#487575" />
<SortedDescendingCellStyle BackColor="#E5E5E5" />
<SortedDescendingHeaderStyle BackColor="#275353" />
                    </asp:GridView></td>
        </table>
</div>
    </form>
</body>
</html>
```

Default.aspx.cs Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        XDocument document = XDocument.Load(@"F:\ASP.NET Rizvi\LINQ
PROG\EMPLOYEE.xml");

        var query = from r in document.Descendants("Employee")
                    select new

                    {

                        FirstName = r.Element("FirstName").Value,
                        Age = r.Element("Age").Value,
                        Dept = r.Element("Dept").Value

                    };

        GridView1.DataSource = query;
```

```
        GridView1.DataBind();

    }
}
```



13.5.3 LINQ to DataSet

The Dataset is a standerd object used in ado.net to work with disconnected data from a variety of data sources and optionally update data source at a later time with changes made working in disconnected mode. Linq to dataset lets you query dataset objects using linq queries.Linq to Dataset also lets you easily and flexible solutions to support tasks such as generic reporting and analysis.

A linq to dataset query is shown below in following example :-

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

namespace LinqToDataset
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectString = "Data Source=HP;" +
                "Integrated security=true;Initial Catalog=Super_old;";
```

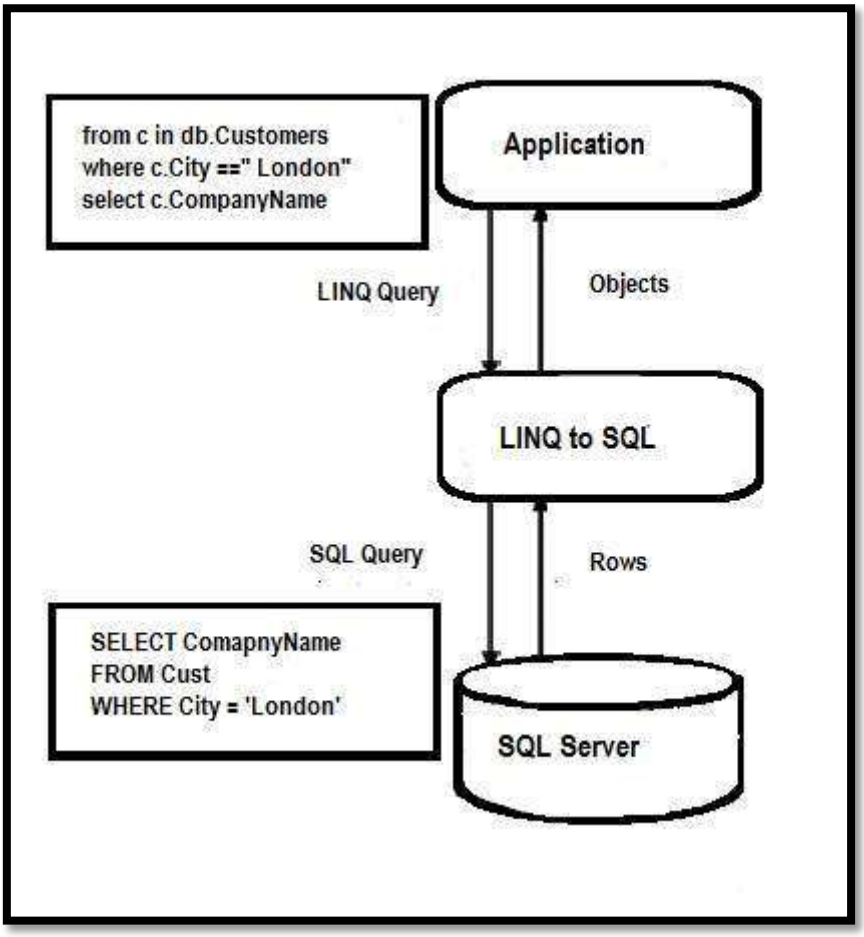
```
            string sqlSelect = "SELECT * FROM mstore;" +
               "SELECT * FROM msalesman;";

            // Create the data adapter to retrieve data from the database
            SqlDataAdapter da = new SqlDataAdapter(sqlSelect, connectString);
            // Create table mappings
            da.TableMappings.Add("Table", "mStore");
            da.TableMappings.Add("Table1", "mSalesMan");
            // Create and fill the DataSet
            DataSet ds = new DataSet();
            da.Fill(ds);

            DataRelation dr = ds.Relations.Add("StoreId_key",
                     ds.Tables["mStore"].Columns["StoreId"],
                     ds.Tables["mSalesMan"].Columns["StoreId"]);

            DataTable Store = ds.Tables["mStore"];
            DataTable SaleMan = ds.Tables["mSalesMan"];

            var query = from p in Store.AsEnumerable()
                        join i in SaleMan.AsEnumerable()
                           on p.Field<int>("StoreId") equals
                              i.Field<int>("StoreId")
                        where p.Field<int>("StoreId") == 2
                        select new
                        {
                           StoreId = p.Field<int>("StoreId"),
                           Name = p.Field<string>("Name"),
                           SalesManName = i.Field<string>("Name")

                        };

            foreach (var q in query)
            {
               Console.WriteLine("StoreId = {0} , StoreName = {1} , SalesManName = {2}",
                  q.StoreId, q.Name, q.SalesManName);
            }

            Console.WriteLine("\nPress any key to continue.");
            Console.ReadKey();
         }
      }
   }
```

13.5.4 LINQ to SQL (DLINQ)

LINQ to SQL offers an infrastructure (run-time) for the management of relational data as objects. It is a component of version 3.5 of the .NET Framework and ably does the translation of language-integrated queries of the object model into SQL. These queries are then sent to

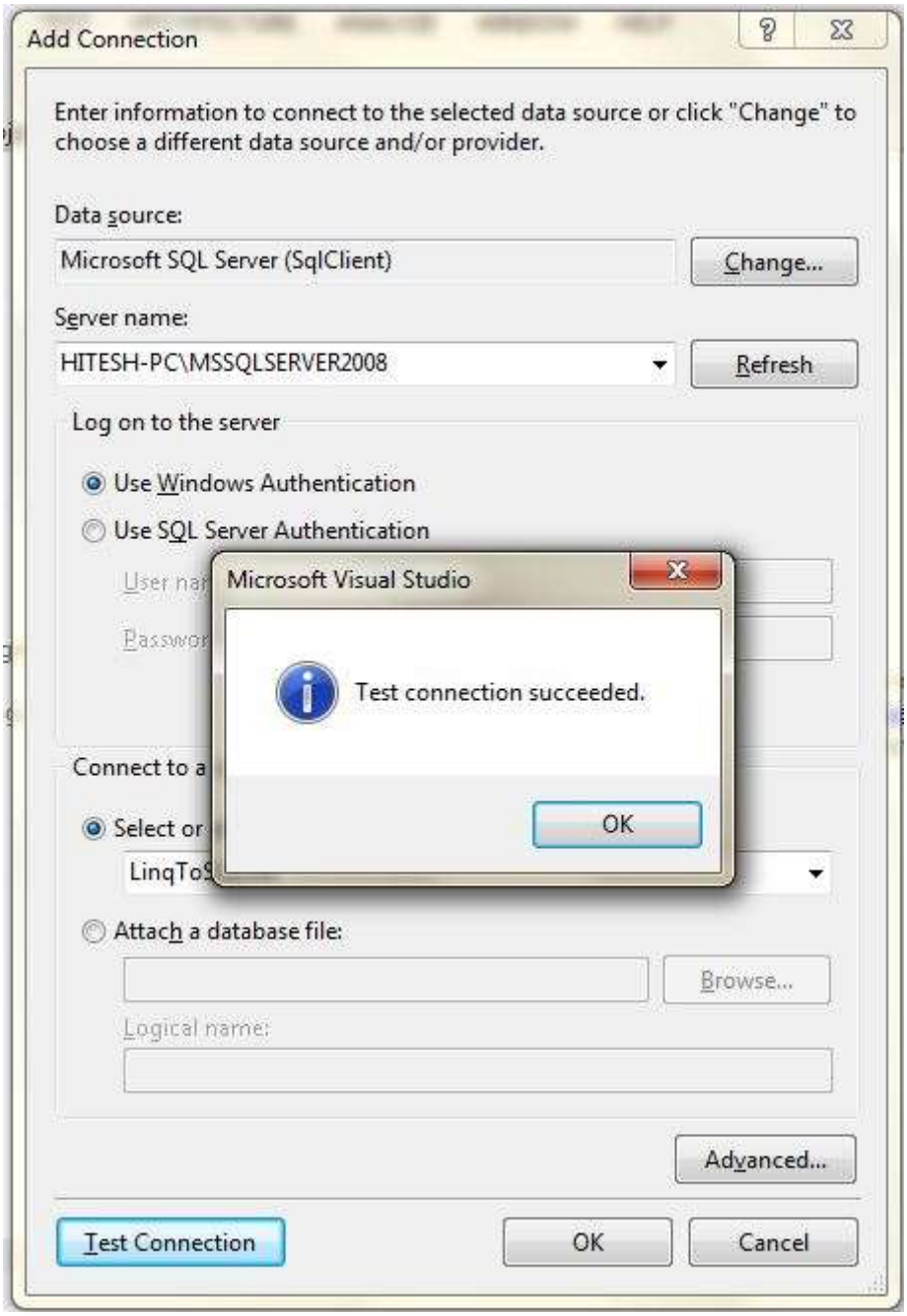
the database for the purpose of execution. After obtaining the results from the database, LINQ to SQL again translates them to objects.

Below is a diagram showing the execution architecture of LINQ to SQL.

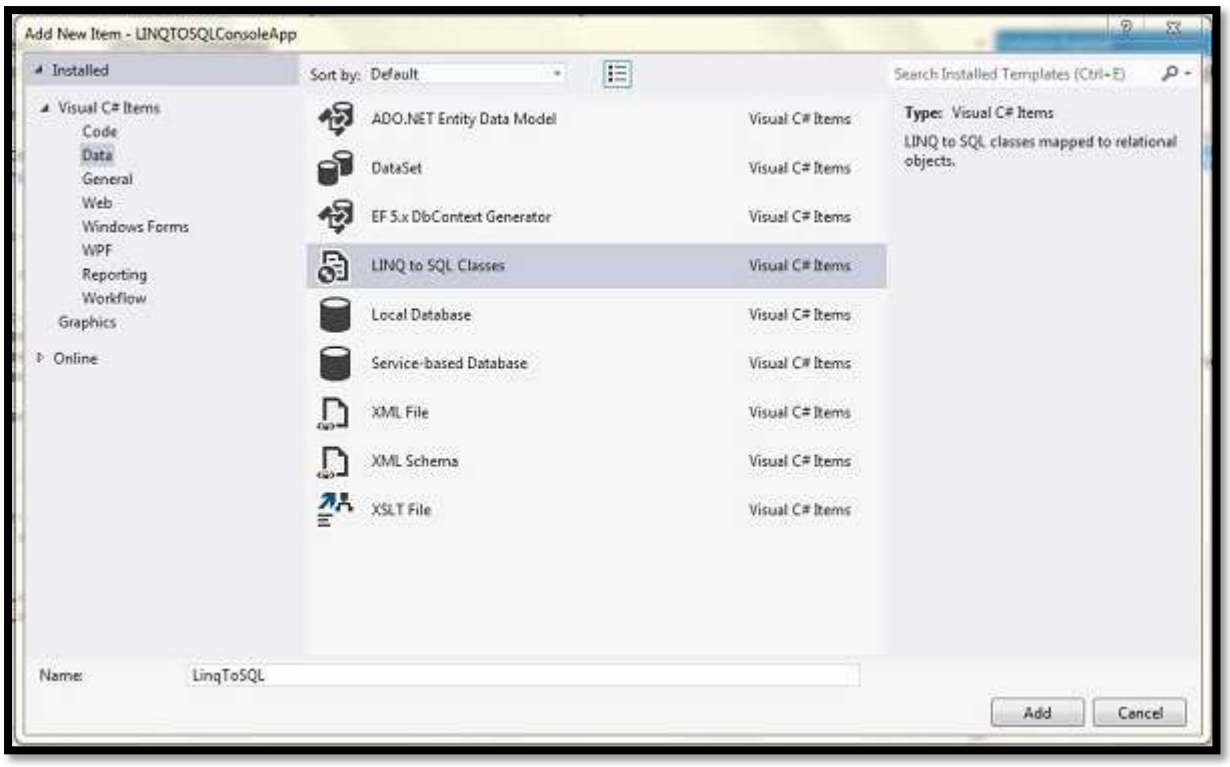


How to Use LINQ to SQL?

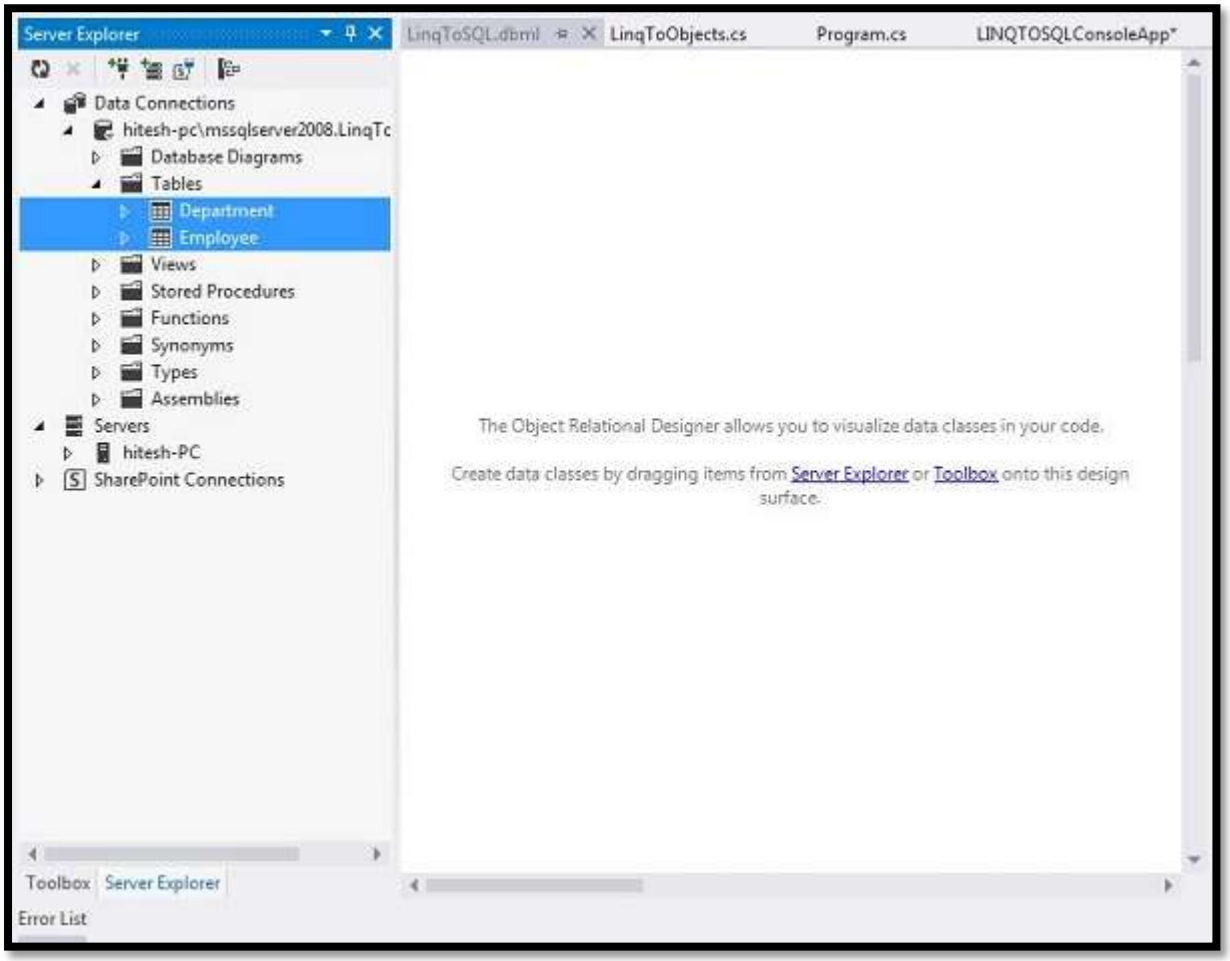
Step 1 − Make a new “Data Connection” with database server. View &arrar; Server Explorer &arrar; Data Connections &arrar; Add Connection



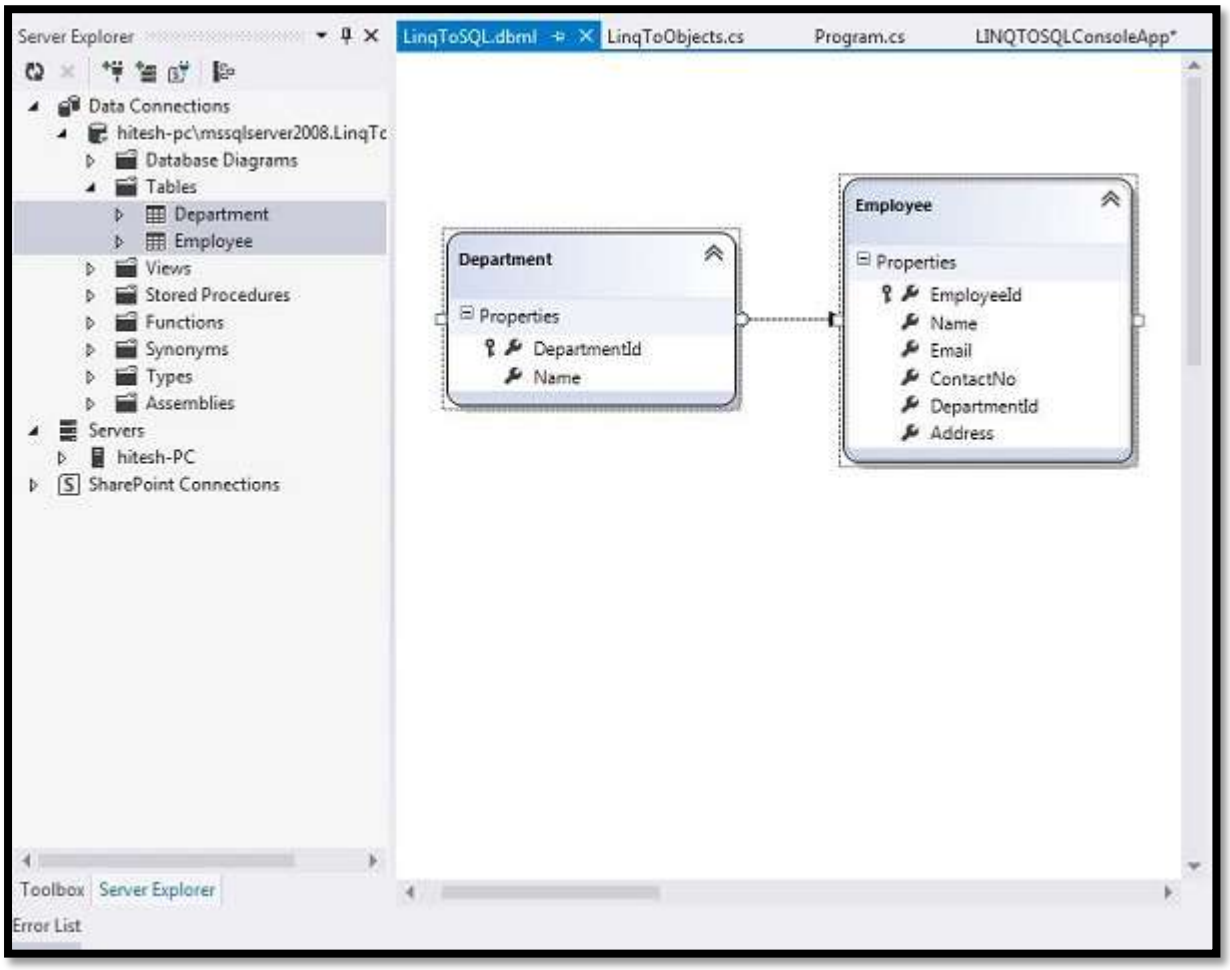
Step 2 − Add LINQ To SQL class file



Step 3 − Select tables from database and drag and drop into the new LINQ to SQL class file.



Step 4 − Added tables to class file.



Insert, Update and Delete using LINQ To SQL

Insert Code:

```
using System;
using System.Linq;

namespace LINQtoSQL {
  class LinqToSQLCRUD {
    static void Main(string[] args) {
      string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["LinqToSQLDBConnection
String"].ToString();

      LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

      //Create new Employee

      Employee newEmployee = new Employee();
      newEmployee.Name = "Michael";
      newEmployee.Email = "yourname@companyname.com";
```

```
        newEmployee.ContactNo = "343434343";
        newEmployee.DepartmentId = 3;
        newEmployee.Address = "Michael - USA";

        //Add new Employee to database
        db.Employees.InsertOnSubmit(newEmployee);

        //Save changes to Database.
        db.SubmitChanges();

        //Get new Inserted Employee
        Employee insertedEmployee =
db.Employees.FirstOrDefault(e⇒e.Name.Equals("Michael"));

        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3},
Address = {4}",
                    insertedEmployee.EmployeeId, insertedEmployee.Name,
insertedEmployee.Email,
                    insertedEmployee.ContactNo, insertedEmployee.Address);

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
      }
   }
}
```

Update Code:

```
using System;
using System.Linq;

namespace LINQtoSQL {
   class LinqToSQLCRUD {
      static void Main(string[] args) {
        string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["LinqToSQLDBConnection
String"].ToString();

        LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

        //Get Employee for update
        Employee employee = db.Employees.FirstOrDefault(e =>e.Name.Equals("Michael"));

        employee.Name = "George Michael";
        employee.Email = "yourname@companyname.com";
        employee.ContactNo = "99999999";
        employee.DepartmentId = 2;
        employee.Address = "Michael George - UK";
```

```
        //Save changes to Database.
        db.SubmitChanges();

        //Get Updated Employee
        Employee updatedEmployee = db.Employees.FirstOrDefault(e
⇒e.Name.Equals("George Michael"));

        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3},
Address = {4}",
                    updatedEmployee.EmployeeId, updatedEmployee.Name,
updatedEmployee.Email,
                    updatedEmployee.ContactNo, updatedEmployee.Address);

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
      }
   }
}
```

Delete Code:

```
using System;
using System.Linq;

namespace LINQtoSQL {
   class LinqToSQLCRUD {
      static void Main(string[] args) {
        string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["LinqToSQLDBConnection
String"].ToString();

        LinqToSQLDataContext db = newLinqToSQLDataContext(connectString);

        //Get Employee to Delete
        Employee deleteEmployee = db.Employees.FirstOrDefault(e ⇒e.Name.Equals("George
Michael"));

        //Delete Employee
        db.Employees.DeleteOnSubmit(deleteEmployee);

        //Save changes to Database.
        db.SubmitChanges();

        //Get All Employee from Database
        var employeeList = db.Employees;
        foreach (Employee employee in employeeList) {
       Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3}",
            employee.EmployeeId, employee.Name, employee.Email, employee.ContactNo);
        }
```

```
        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
      }
    }
}
```

13.5.5 LINQ to ADO.NET

The LINQ to ADO.NET means using LINQ queries on objects in ADO.NET. The LINQ to ADO.NET will give us a chance to write LINQ queries on enumerable object in ADO.NET and the LINQ to ADO.NET is having three types of LINQ technologies available those are LINQ to Dataset, LINQ to SQL and LINQ to Entities.

Let see an example of LINQ to ADO.NET.

Create one new web application and make connection with SQL Server and write queries on ADO.NET object (**dataset**) using LINQ to display data in gridview.

First we will create one new table "**EmployeeDetails**" in database for that execute following query in your database and insert some dummy data to show it in application.

```
CREATE TABLE [dbo].[EmployeeDetails](
[EmpId]  INT IDENTITY (1, 1) NOT NULL,
[EmpName]  VARCHAR (50) NULL,
[Location] VARCHAR (50) NULL,
[Gender] VARCHAR (20) NULL
PRIMARY KEY CLUSTERED ([EmpId] ASC)
);

insert into EmployeeDetails values ('Suresh Dasari','Chennai','Male')
insert into EmployeeDetails values ('Rohini Alavala','Chennai','Female')
insert into EmployeeDetails values ('Praveen Alavala','Guntur','Male')
insert into EmployeeDetails values ('Sateesh Chandra','Vizag','Male')
insert into EmployeeDetails values ('Sushmitha','Vizag','Female')
```

Once we select new project new popup will open in that select **Asp.Net Empty Web Application** and give name as "**LINQtoADONET**" and click **OK** to create new web application.

Now we will add web page to the application for that Right click on your application --> Select Add --> New Item --> Select Web Form --> give name as "**Default.aspx**" and click **OK** button it will create new page in application.

Default.aspx page code:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1">
<title>Bind Gridview with LINQ to ADO.NET Operations</title>
<style type="text/css">
.GridviewDiv {font-size: 100%; font-family: 'Lucida Grande', 'Lucida Sans Unicode',
Verdana, Arial, Helevetica, sans-serif; color: #303933;}
.headerstyle
{
color:#FFFFFF;border-right-color:#abb079;border-bottom-color:#abb079;background-color:
#df5015;padding:0.5em 0.5em 0.5em 0.5em;text-align:center;
}
</style>
</head>
<body>
<form id="form1" runat="server">
<div class="GridviewDiv">
<asp:GridView ID="gvDetails" CssClass="Gridview" runat="server"
AutoGenerateColumns="False">
<HeaderStyle CssClass="headerstyle" />
<Columns>
<asp:BoundField HeaderText="Name" DataField="Name" />
<asp:BoundField HeaderText="Location" DataField="Location" />
<asp:BoundField HeaderText="Gender" DataField="Gender" />
</Columns>
</asp:GridView>
</div>
</form>
</body>
</html>
```

Default.aspx.cs page code:

```
using System;
using System.Web.UI;
using System.Data.SqlClient;
using System.Data;

namespace WebApplication2
{
public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if (!Page.IsPostBack)
{
BindGridview();
}
}
```

```
protected void BindGridview()
{
DataSet ds = new DataSet();
using (SqlConnection con = new SqlConnection("Data
Source=HP\\SQLEXPRESS;Integrated Security=true;Initial Catalog=MySampleDB"))
{
con.Open();
SqlCommand cmd = new SqlCommand("select * from employeedetails", con);
cmd.CommandType = CommandType.Text;
SqlDataAdapter da = new SqlDataAdapter(cmd);
da.Fill(ds);
con.Close();
if (ds.Tables[0].Rows.Count > 0)
{
var result = from dt in ds.Tables[0].AsEnumerable()
where (dt.Field<string>("Gender") == "Male")
select new
{
Name = dt.Field<string>("EmpName"),
Location = dt.Field<string>("Location"),
Gender = dt.Field<string>("Gender"),
};
gvDetails.DataSource = result;
gvDetails.DataBind();
}
}
}
}
}
```

13.6 SUMMARY

This unit gives an overview of LINQ and its operation in ASP.NET.

13.7 EXERCISE

1. Explain the LINQ query syntax with an example query.
2. Differentiate between SQL and LINQ.
3. Write advantages and disadvantages of LINQ.
4. Short Note on LINQ to object and Query Expression.
5. Explain the query operators SELECT, FROM, ORDERBY and WHERE in LINQ.
6. Explain LINQ to XML with example.
7. Explain LINQ to SQL with example.
8. Explain LINQ to ADO.NET with example.

REFERENCE

1. ADO.NET: The Complete Reference
2. Introducing Microsoft LINQ by Paolo Pialorsi

3. Database Programming with Visual Basic .Net and ADO.NET by F Scott Barker, Publisher: Pearson Education
4. LINQ Unleashed for C# Book by Paul Kimmel
5. https://www.c-sharpcorner.com/
6. https://stackoverflow.com/
7. https://www.tutorialspoint.com
8. http://www.tutorialsteacher.com/linq/linq-tutorials

ASP.NET SECURITY

14.0 INTRODUCTION

When you begin a program for a customer using ASP.Net, you should consider about security. Security is one of the most important components of any application. Security is even more important when you are making a web application which is exposed to million of users. Asp.net provides classes and methods that ensure that the application is secure from outside attacks.

ASP.NET provides an extensive security model that makes it easy to protect your web applications. Although this security model is powerful and profoundly flexible, it can appear confusing because of the many different layers that it includes. Much of the work in securing your application is not writing code, but determining the appropriate places to implement your security strategy.

14.1 AUTHENTICATION

- If you want to limit access to all or part of your ASP.NET application to certain users, you can use authentication to verify each user's identity.
- Then, once you have authenticated the user, you can use authorization to check if the user has the appropriate privileges for accessing a page.
- Authentication refers to the process of validating the identity of a user so the user can be granted access to an application. A user must typically supply a user name and password to be authenticated.

- After a user is authenticated, the user must still be authorized to use the requested application. The process of granting user access to an application is called authorization.

14.1.1 Three types of authentication

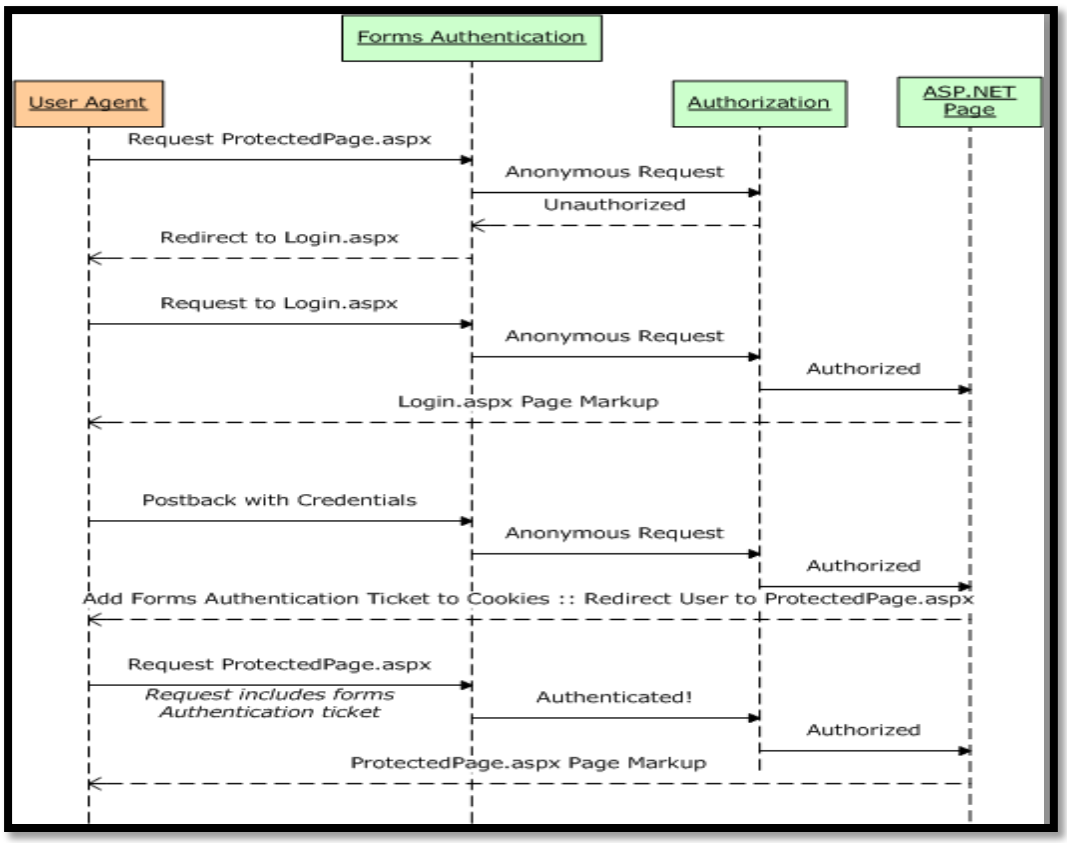
1. **Windows-based authentication**
   - Causes the browser to display a login dialog box when the user attempts to access a restricted page.
   - Is supported by most browsers.
   - Is configured through the IIS management console.
   - Uses Windows user accounts and directory rights to grant access to restricted pages.

2. **Forms-based authentication**
   - Lets developers code a login form that gets the user name and password.
   - The user name and password entered by the user are encrypted if the login page uses a secure connection.
   - Doesn't rely on Windows user accounts. Instead, the application determines how to authenticate users.
3. **Windows Live ID authentication**
   - Windows Live ID is a centralized authentication service offered by Microsoft.
   - Windows Live ID lets users maintain a single user account that lets them access any web site that participates in Windows Live ID.
   - The advantage is that the user only has to maintain one user name and password.
   - To use Windows Live ID, you must register your web site with Microsoft to obtain an application ID and then download the Windows Live ID Web Authentication SDK.

- To start the ASP.NET Web Site Administration Tool, use the Website→ASPNET Configuration command.
- You can use the ASP.NET Web Site Administration Tool to set up users, roles, and access rules.

How to enable forms-based authentication

- By default, a web site is set up to use Windows authentication. However, this option is only appropriate for accessing a web site through a local intranet.
- To switch to forms-based authentication, select the From the Internet option. This is the option that you'll need to use if you intend to deploy the application so it's available from the Internet.
- When you switch to forms-based authentication, Create User and Manage Users links become available from the main security page.



Forms-Based Authentication:

Traditionally forms based authentication involves editing the Web.Config file and adding a login page with appropriate authentication code.

The Web.Config file could be edited and the following codes written on it:

```
<system.web>

<authentication mode="Forms">
    <forms loginUrl ="login.aspx"/>
</authentication>
<authorization>
    <deny users="?"/>
 </authorization>

</system.web>
...
...
</configuration>
```

The login.aspx page mentioned in the above code snippet could have the following code behind file with the usernames and passwords for authentication hard coded into it.

```
protected bool authenticate(String uname, String pass)
{
  if(uname == "Tom")
  {
    if(pass == "tom123")
      return true;
```

```
  }
  if(uname == "Dick")
  {
    if(pass == "dick123")
      return true;
  }
  if(uname == "Harry")
  {
    if(pass == "har123")
      return true;
  }
  return false;
}

public void OnLogin(Object src, EventArgs e)
{
  if (authenticate(txtuser.Text, txtpwd.Text))
  {
    FormsAuthentication.RedirectFromLoginPage(txtuser.Text,
                                chkrem.Checked);
  }
  else
  {
    Response.Write("Invalid user name or password");
  }
}
```

Observe that the FormsAuthentication class is responsible for the process of authentication.
However, Visual Studio allows you to implement user creation, authentication and authorization with seamless ease without writing any code, through the Web Site Administration tool. This tool allows creating users and roles.
Apart from this, ASP.Net comes with readymade login controls set, which has controls performing all the **jobs for you.**

14.2 AUTHORIZATION

After your application has authenticated users, you can proceed to authorize their access to resources. But there is a question to answer first: Just who is the user to whom your are grating access? It turns out that there are different answers to that question, depending on whether you implement impersonation. Impersonation is a technique that allows the ASP.NET process to act as the authenticated user, or as an arbitrary specified user

14.3 IMPLEMENTING FORMS-BASED SECURITY

To set up forms based authentication, the following things are needed:

- A database of users to support the authentication process
- A website that uses the database
- User accounts
- Roles
- Restriction of users' and group activities

You need:

- A default page, which will display the login status of the users and other information
- A login page, which will allow users to log in, retrieve password or change password

To create users take the following steps:

Step (1): Choose Website -> ASP.Net Configuration to open the Web Application Administration Tool

Step (2) : Click on the Security tab:

Step (3): Select the authentication type to Forms based authentication by selecting the 'From the Internet' radio button.

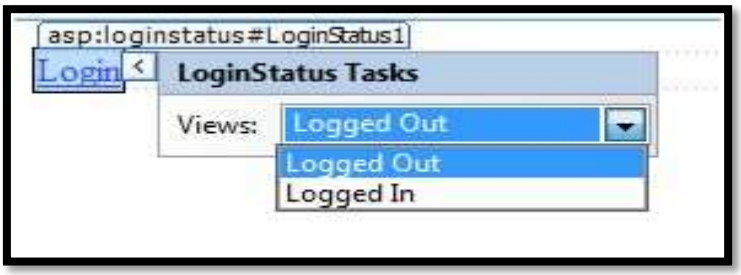
Step (4): Click on 'Create Users' link to create some users. If you already had created roles, you could assign roles to the user, right at this stage.

Step (5): Create a web site and add the following pages:

- Welcome.aspx
- Login.aspx
- CreateAccount.aspx
- PasswordRecovery.aspx
- ChangePassword.aspx

Step (6) : Place a LoginStatus control on the Welcome.aspx from the login section of the toolbox. It has the templates: Logged in and Logged out.

In Logged out template, there is a login link and in the Logged in template, there is a logout link on the control. You can change the login and logout text properties of the control from the Properties window.

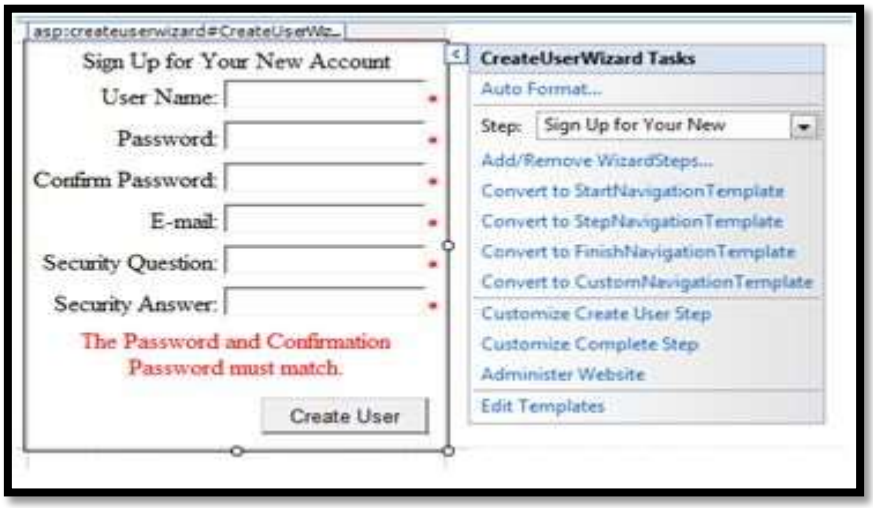


Step (7): Place a LoginView control from the toolbox below the LoginStatus control. Here you can put texts and other controls (hyperlinks, buttons etc), that will be displayed based on whether the user is logged in or not.

This control has two view templates: Anonymous template and Logged in template. Select each view and write some text for the users to be displayed for each template. The text should be placed on the area marked red.

Step (8): The users for the application are created by the developer. You might want to allow a visitor to the site create a user account. For this, add a link beneath the LoginView control, which should link to the CreateAccount.aspx page.

Step (9): Place a CreateUserWizard control on the create account page. Set the ContinueDestinationPageUrl property of this control to Welcome.aspx.



Step (10): Create the Login page. Place a Login control on the page. The LoginStatus control automatically links to the Login.aspx. To change this default, make the following changes in the web.config file.

For example, if you want to name your log in page as signup.aspx, add the following lines to the <authentication> section of the web.config:

```
<authentication mode="Forms">
    <forms loginUrl ="signup.aspx"
          defaultUrl = “Welcome.aspx” />
</authentication>
</system.web>
</configuration>
```

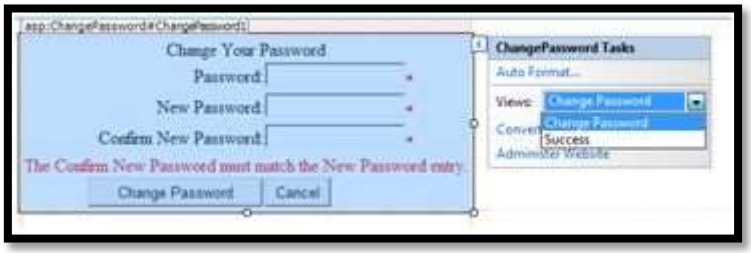
Step (11): Users often forget passwords. The PasswordRecovery control helps the user gain access to the account. Select the Login control. Open its smart tag and click ‘Convert to Template’.

Customize the UI of the control to place a hyperlink control under the login button, which should link to the PassWordRecovery.aspx.

Step (12): Place a PasswordRecovery control on the password recovery page. This control needs an email server to send the passwords to the users.

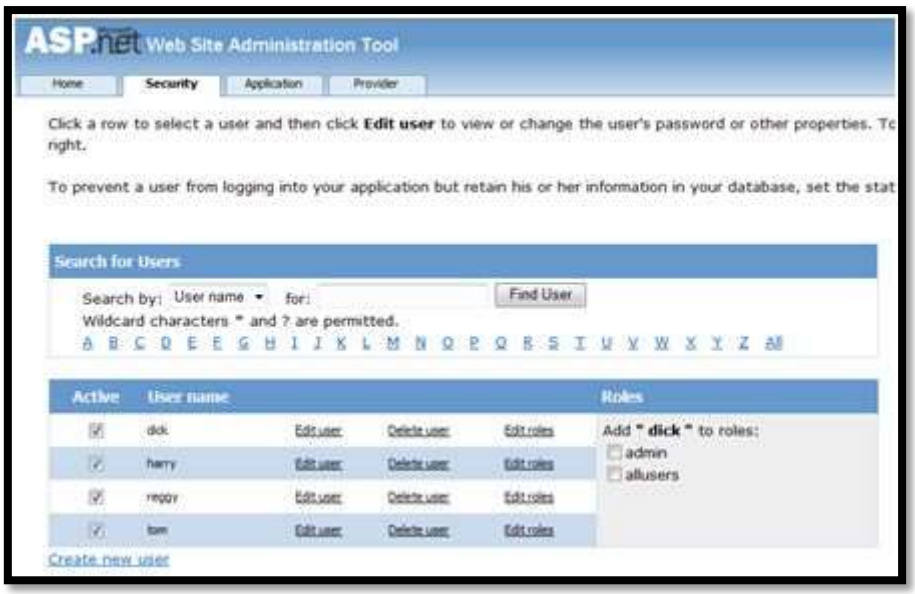
Step (13): Create a link to the ChangePassword.aspx page in the LoggedIn template of the LoginView control in Welcome.aspx.

Step (14): Place a ChangePassword control on the change password page. This control also has two views.



Now run the application and observe different security operations.
To create roles, go back to the Web Application Administration Tools and click on the Security tab. Click on 'Create Roles' and crate some roles for the application.

Click on the 'Manage Users' link and assign roles to the users.



14.4 FORM AUTHENTICATION STEPS IN ASP.NET

1. Create a new blank web site.
2. From the **Website** menu, select **ASP.NET Configuration.**
3. The **Web Site Administration Tools (WAST)** opens.
4. Go the Security Tab .Click **Select authentication type.**
5. Select the option button **From The Internet.**
6. Click **Back** button to the main security tab.
7. Click **Enable roles.**
8. Click Create or **Manage roles**.
9. Add two roles: **Webmaster** and **Webuser**. Roles are like groups. They provide an easy way to assign permissions.
10. After creating the two roles, go **Back** to the main security tab.
11. Then go to user tab on **WAST**.
12. The first user will be added to the Webmaster role. Fill in the form with appropriate information. Create to user account.
13. Select role for each user.
14. Click **Create User** button.
15. Add two folder in solution explorer give the name **Secure** and **Public.**
16. Add two forms to the web site **MainLogin.aspx** and change **Password.aspx.**
17. Add one **web page** in to **secure folder**. Rename it **secure.aspx.**
18. Add one **web page** in to **Public folder**. Rename it **public.aspx**
19. Go to **MainLogin.aspx** page and add a **Login Control**.
20. Go to **ChangePassword.aspx** page and add a **ChangePassword Control**.
21. Change the property of **ChangePassword** control **SussessPageUrl** to the public page.
22. Put some text on the **public page** and **secure page**.
23. Open **web.config** file modify the **authentication** setting inside the system.web section.
24. Insert the code <forms **loginurl** = **"MainLogin.aspx "** >.
25. Build the website.
26. Go to **WAST → main security tab→ click create access rules**.
27. Create access rules for the folders you created. These determine which roles are allowed to see the pages.
28. Click the Public folder and **deny Anonymous user**.
29. Create another rule on public to allow all users. This allowed authenticated users only.

30. Next repeat these steps on the secure folder, but deny Anonymous and all users.
31. After allowing Webmaster access to the secure folder, check it by clicking **Manage access rules**.
32. Click on done button.
33. View the secure page in the browser.
34. Login as a Masooma.....Nothing will happen.
35. Then Login as Zaidi.....Secure Page will display.

14.5 IMPERSONATION

When using impersonation, ASP.NET applications can execute with the Windows identity (user account) of the user making the request. Impersonation is commonly used in applications that rely on Microsoft Internet Information Services (IIS) to authenticate the user.

One of our websites uses Impersonation and a specific user account with special permissions to access certain system resources. The first step in enabling impersonation is setting up the correct attributes in the *web.config* file:

```
<system.web>
<identity impersonate="true" password="xxxxxx" userName="xxxxxxx" />
```

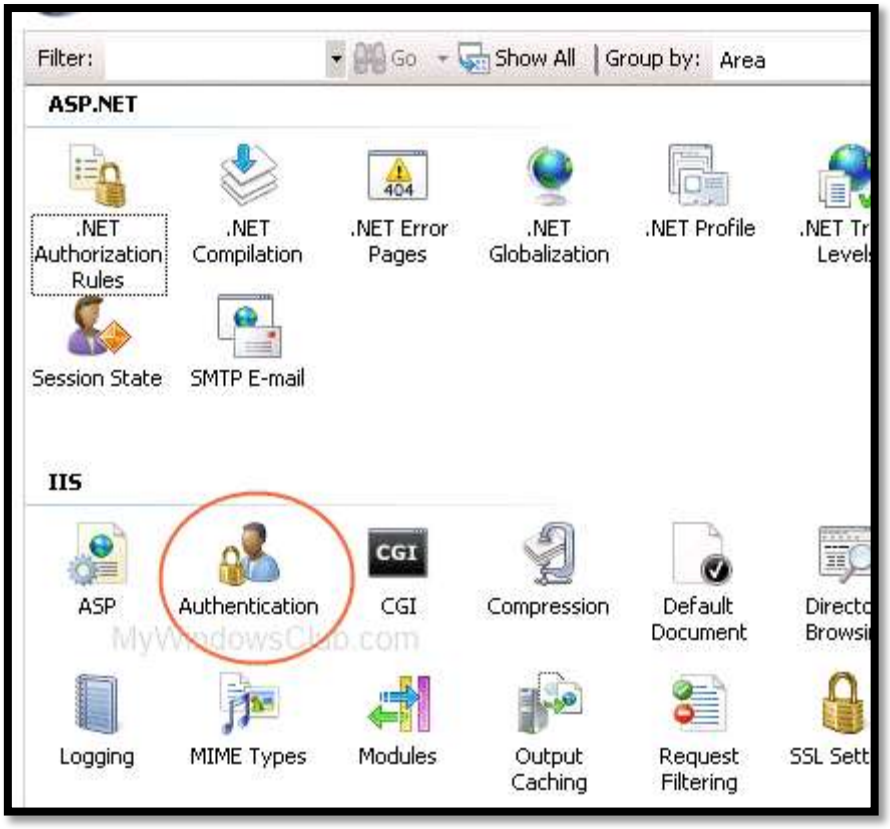
By using the attribute impersonate="true", you are telling IIS that this website will be impersonating the configured user account.

Configure the website to use a specific user account

The next step is you need to go to IIS Manager and configure the user account you want to impersonate by this website.

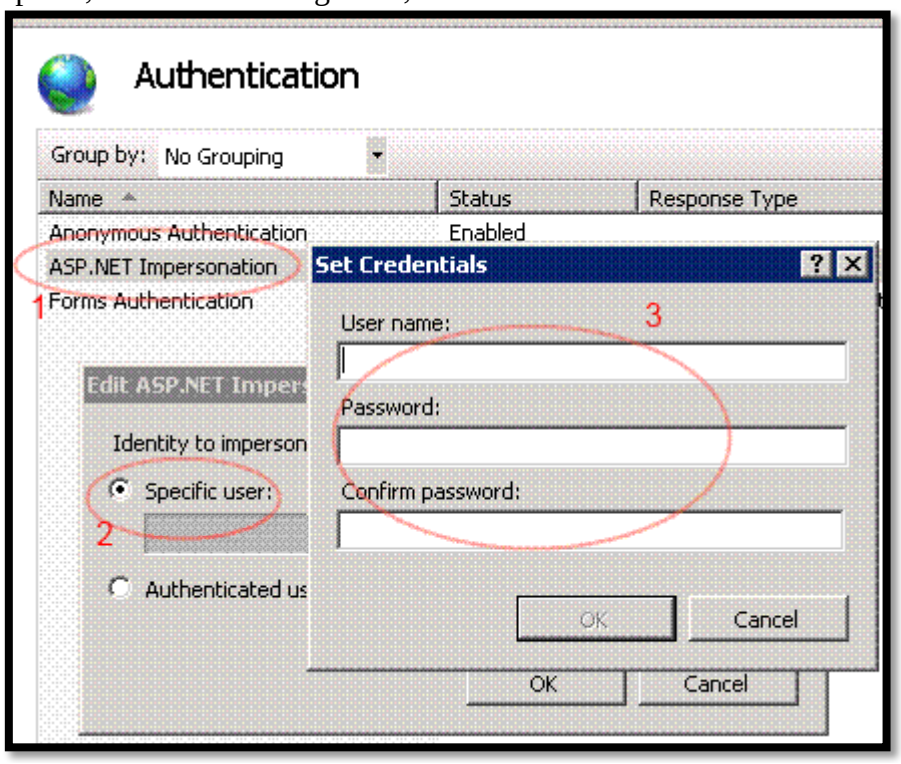
Steps

1. Open IIS Manager.



2. Expand computer name.

3. Expand websites.
4. Click on the specific website for which you want to use impersonation.
5. On the right panel, under the heading "IIS", double click "Authentication".



6. Right click on "ASP.NET Impersonation" and select "Edit".
7. Choose "Specific User".
8. Click the SET button to provide the specific user name and password.
   Press OK at the popup dialog to complete this step on enabling impersonation for website in IIS 7.0.

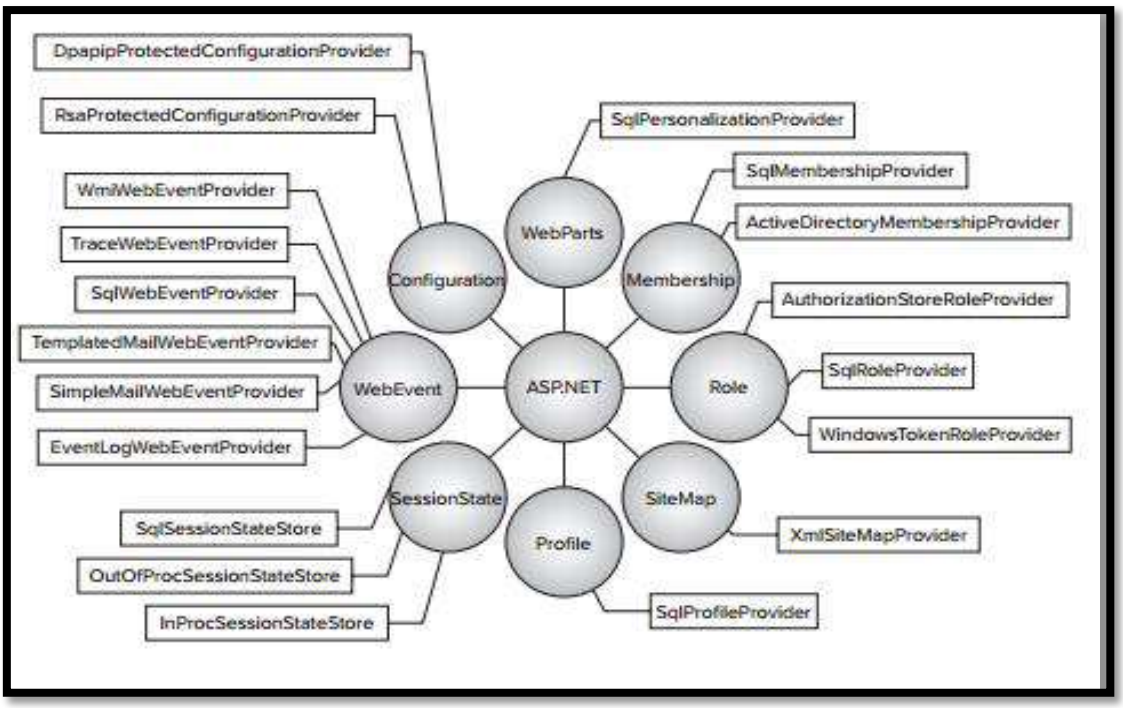
14.6 ASP.NET PROVIDER MODEL

ASP.NET includes a number of services that store state in databases and other storage media.

A provider is a software module that provides a uniform interface between a service and a data source. Providers abstract physical storage media, in much the same way that device drivers abstract physical hardware devices.

The following are the built-in providers of the ASP.NET Provider Model.

- Membership Providers
- Role Management Providers
- Profile Providers
- Site Map Providers
- Session State Providers
- Web Event Providers
- Web Parts Personalization Providers
- Protected Configuration Providers



Goals of the Provider Model

The ASP.NET 2.0 provider model was designed with the following goals in mind:

- To make ASP.NET state storage both flexible and extensible.
- To insulate application-level code and code in the ASP.NET run-time from the physical storage media where state is stored, and to isolate the changes required to use alternative media types to a single well-defined layer with minimal surface area.
- To make writing custom providers as simple as possible by providing a robust and well-documented set of base classes from which developers can derive provider classes of their own.

Provider Types

- Membership is one of several ASP.NET services that use the provider architecture. The following table documents the features and services that are provider-based and the default providers that service them:

Feature or Service	Default Provider
Membership	System.Web.Security.SqlMembershipProvider
Role management	System.Web.Security.SqlRoleProvider
Site map	System.Web.XmlSiteMapProvider
Profile	System.Web.Profile.SqlProfileProvider
Session state	System.Web.SessionState.InProcSessionStateStore
Web events	N/A (see below)
Web Parts personalization	System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider
Protected configuration	N/A (see below)

14.7 SUMMARY

This unit gives an overview of security in ASP.NET such as authentication, authorization and impersonation. Further it discusses about provider model.

14.8 EXERCISE

1. Explain the term authentication with respect to ASP.NET security.
2. What is the difference between authorisation and impersonation in terms of security in ASP.NET?
3. Steps to use Windows authentication with sample code.
4. ASP.NET Provider model with diagram.
5. What do you mean by authentication? Explain its types.

REFERENCE

1. Beginning ASP.NET 4.5 in C# by Matthew MacDonald
2. Beginning ASP.NET Security by Barry Dorrans
3. https://www.c-sharpcorner.com/
4. https://stackoverflow.com/
5. https://support.microsoft.com/en-us/help/891028/asp-net-security-overview

ASP.NET AJAX

Content

15.0 INTRODUCTION

Ajax, shorthand for Asynchronous JavaScript and XML. In other words Ajax is the combination of various technologies such as a JavaScript, CSS, XHTML, and DOM. The first known use of the term in public was by Jesse James Garrett in his February 2005 article Ajax: A New Approach to Web Applications.

ASP.NET AJAX, previously called "Atlas", is a Microsoft implementation of an AJAX based framework, created for ASP.NET. This allows for a richer experience for the user, since loading dynamic content can be done in the background, without refreshing and redrawing the entire page. Google have made Ajax very popular.

15.1 ADVANTAGES AND DISADVANTAGES OF AJAX

Advantages of AJAX

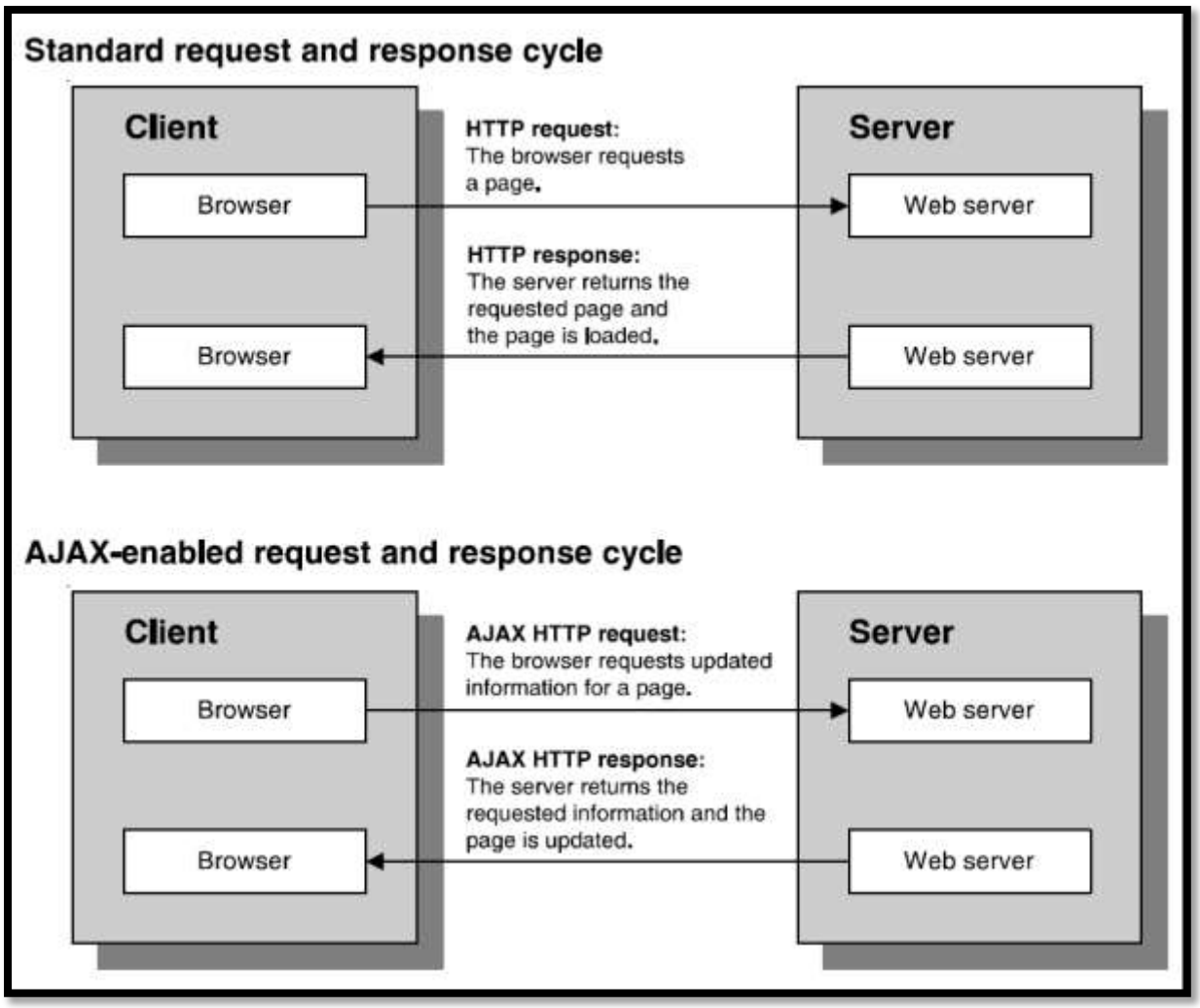
- When using Ajax, a web application can request only the content that needs to be updated, thus drastically reducing bandwidth usage and load time.
- The web application will be operated faster or more responsive, even if the application has not changed on the server side.
- Ajax enable to reduce connections to the server, since scripts and style sheets only have to be requested once.
- State can be maintained throughout a Web site such as JavaScript variables.
- AJAX enables a much better user experience for Web sites and applications.
- Developers can now provide user interfaces that are nearly as responsive and rich as more traditional Windows Forms applications while taking advantage of the Web's innate ease of deployment and heterogeneous, cross-platform nature.
- These benefits have been shown to dramatically reduce software maintenance costs and increase its reach. You can use AJAX to load specific portions of a page that need to be changed.
- It further reduces network traffic.

Disadvantages of AJAX

- ActiveX requests are enabled only in IE 5 and IE6
- AJAX is not well integrated with any browser.
- Clicking the browser's "back" button may not return the user to an earlier state of the Ajax-enabled page.
- Dynamic web page updates also caused some troubles for a user to bookmark a particular state of the application.
- Ajax opens up another attack vector for malicious code that web developers might not expected for.
- Any user whose browser does not support Ajax or JavaScript, or simply has JavaScript disabled, will not be able to use its functionality.

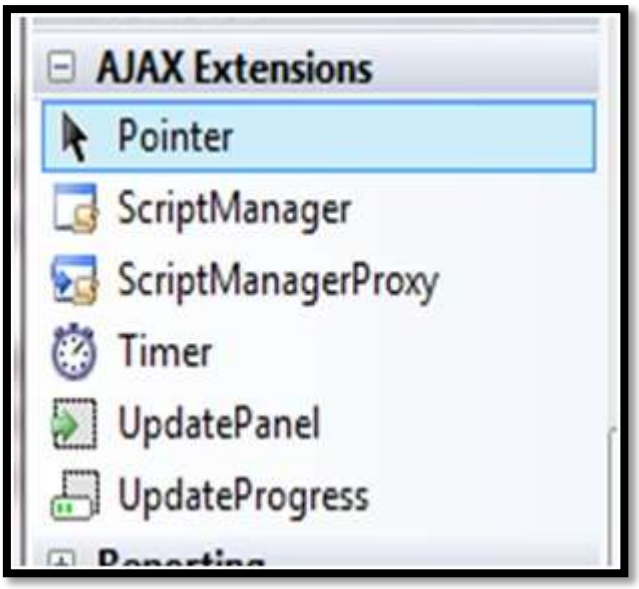
15.2 PARTIAL REFRESHES

- The key technique in an Ajax web application is partial refreshes. With partial refreshes, the entire page doesn't need to be posted back and refreshed in the browser.
- Instead, when something happens the web page asks the web server for more information. The request takes place in the background, so the web page remains responsive.



15.3 ASP.NET AJAX CONTROL TOOLKIT

- Ajax Control Toolkit is an open source library for web development.
- The ASP.net Ajax Control toolkit contains highly rich web development controls for creating responsive and interactive AJAX enabled web applications.
- Controls are available in the Visual Studio Toolbox for easy drag and drop integration with your web application.
- AJAX Extension supports the .NET Framework to build high quality application consisting client side scripts.
- In order to improve the web applications in nature of the AJAX architecture developers prefer it.
- .NET Framework has list of controls based on AJAX Extension.
- The toolbox of the asp.net in visual studio contains a group of Ajax Extender as shown below:



15.3.1 THE POINTER CONTROL

It is just a pointer. If we drag any other control on form it causes to create that control on form but pointer does not create any control on form. In other word we can say, we select it for to ignore any other selected control.

15.3.2 THE SCRIPTMANAGER CONTROL

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

<asp:ScriptManager ID="ScriptManager1" runat="server"> </asp:ScriptManager>

More about ScriptManager Control

- Downloads JavaScript files to client.
- Enables partial-page rendering using UpdatePanel.
- Provides access to Web services via client-side proxies.
- Manages callback timeouts and provides error handling options and infrastructure
- Provides registration methods for scripts.
- Enables ASP.NET AJAX localization support.
- Every page requires one ScriptManager instance!

15.3.3 THE SCRIPTMANAGERPROXY CONTROL

- This control is used on content page when we have ScriptManager on Master page. From content page ScriptManagerProxy hooks itself to ScriptManager at runtime.

15.3.4 THE UPDATEPANEL CONTROL

- Enable sections of a page to be partially rendered without a post back called as Partial page rendering.
    - Clean round trips to server and flicker-free updates.
    - Requires no knowledge of JavaScript or AJAX.
- A declarative model that works like ASP.NET server controls.
- Can be used with Master pages, User Controls and Data Bound Controls.
- Single or Multiple Update panel controls can also be used on a Web page.
- It can be created and refreshed programmatically.

It has the basic syntax:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server"> </asp:UpdatePanel>
```

Properties of the UpdatePanel Control

The properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.
RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

Example

UpdatePanel.aspx Code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UpdatePanel.aspx.cs"
Inherits="UpdatePanel" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <table class="style1">
         <tr>
         <h1 align="center" UpdatePanel Example</h1>
            </td>
         </tr>
            <td class="style2">
               <asp:ScriptManager ID="ScriptManager1" runat="server">
               </asp:ScriptManager>
            </td>
            <td class="style2">
               <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Names="Goudy
Stout" Font-Size="XX-Large" ForeColor="#000066" Text="TIME "></asp:Label>
            </td>
        <td class="style2">
               <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Names="Goudy
Stout" Font-Size="XX-Large" ForeColor="Red" Text="TIME"></asp:Label>
            </td>
            <td class="style6">
               <asp:Button ID="Button3" runat="server" Font-Bold="True" Font-
Names="Algerian" Font-Size="XX-Large" ForeColor="#003300" Height="46px"
onclick="Button3_Click" Text="Update Time" Width="222px" />
            </td>
            <td class="style6">
               <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                  <ContentTemplate>
                     <table class="style3" bgcolor="#00FF99" border="5"> <tr><td>
<asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Names="Goudy Stout"
Font-Size="XX-Large" ForeColor="#000066" Text="TIME "></asp:Label> </td> <td>
<asp:Button ID="Button2" runat="server" ClientIDMode="AutoID" Font-Bold="True"
  Font-Names="Algerian" Font-Size="XX-Large" ForeColor="#000099"
  onclick="Button1_Click" Text="UPDATE TIME " Width="237px" />
  </td>
  </table>
  </ContentTemplate>
  </asp:UpdatePanel>
      </table>
   </div>
   </form>
```

```
</body>
</html>
```

UpdatePanel.aspx.cs Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class UpdatePanel : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToLongTimeString();
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label3.Text = DateTime.Now.ToLongTimeString();
    }
    protected void Button3_Click(object sender, EventArgs e)
    {
        Label2.Text = DateTime.Now.ToLongTimeString();
    }
}
```

15.3.5 THE UPDATEPROGRESS CONTROL

- The UpdateProgress Control is used to show the progress of the partial page contents in the UpdatePanel.
- This control is very useful when the speed of updating content is slow.
- Thus a user gets idea how much information is processed and how long a user has to wait.

It has the basic syntax:

```
<asp: UpdateProgress ID="UpdateProgress1" runat="server"> </asp: UpdateProgress>
```

More About UpdateProgress control

- It provides status information about partial-page updates in UpdatePanel controls.
- It helps to prevent flashing when a partial-page update is very fast.
- It helps to design a more intuitive UI when a Web page contains oneor more UpdatePanel controls.
- Displays custom template-driven UI for:
  - Indicating that an async update is in progress.
  - Canceling an async update that is in progress.
- Automatically displayed when update begins or "DisplayAfter" interval elapses.

Properties of the UpdateProgress Control

The properties of the update progress control:

Properties	Description
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.
Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.
DynamicLayout	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The methods of the update progress control:

Methods	Description
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

Example

UpdateProgress.aspx code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UpdateProgress.aspx.cs"
Inherits="UpdateProgress" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <table class="style1">
            <td class="style11">
               <h1>Ajax UpdateProgress Example</h1></td>
            <td class="style10" style="text-align: center">
               <asp:ScriptManager ID="ScriptManager1" runat="server">
               </asp:ScriptManager></td>
            <td class="style10">
               <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                  <ProgressTemplate>
                     <table class="style3">
                        <tr>
<asp:Image ID="Image2" runat="server" Height="179px" ImageUrl="~/Loading1.gif"
Width="250px" /></td>
</table>
                  </ProgressTemplate>
               </asp:UpdateProgress>
            </td>
            <td class="style10">
               <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                  <ContentTemplate>
                     <table class="style3">
                        <tr>
                           <td class="style5">
                              <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
Names="Algerian" Font-Size="XX-Large" Text="Label"></asp:Label>    </td>
                           <td class="style6">
<asp:Button ID="Button1" runat="server" Font-Bold="True" Font-Italic="True"
Font-Names="Arial Black" Font-Size="XX-Large" onclick="Button1_Click"
Text="Update Time" Width="256px" />
</td></tr></table>
                  </ContentTemplate>
               </asp:UpdatePanel>
            </td>
      </table>
   </div>
   </form>
```

```
</body>
</html>
```

UpdateProgress.aspx.cs code:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class UpdateProgress : System.Web.UI.Page
{
 protected void Button1_Click(object sender, EventArgs e)
   {
      System.Threading.Thread.Sleep(5000);
      Label1.Text = DateTime.Now.ToLongTimeString();
   }
}
```

15.3.6 THE TIMER CONTROL

- The Timer Control is used to update the content of a page after a specific interval.
- The Timer Control is attached with the UpdatePanel control to perform the regular updates for the partial page after a predefine interval.
- A developer can add one or more than one control on a single webpage.
- This control is invisible at runtime just like ScriptManager control.

It has the basic syntax:

<asp:Timer ID="Timer1" runat="server"> </asp:Timer>

Some important properties of Timer Control are given below :

- **Interval**: It specifies the desire time limit in milliseconds (1000 milliseconds in a single second). The Tick event is raised after the interval time limit is over.
- **Enabled**: It sets a value that indicates that the Timer Control fires the tick event or not.

Example

Timer.aspx code:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Timer.aspx.cs"
Inherits="Timer" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <table class="style1" bgcolor="#FFFFCC" >
            <td>
               <h1>Timer Control Examample</h1></td>
            <td>
            <td>
               <asp:ScriptManager ID="ScriptManager1" runat="server">
               </asp:ScriptManager>
            </td>
            <td bgcolor="#CCFFCC">
               <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
Names="Algerian"
                  Font-Size="XX-Large" ForeColor="Red" Text="Label"></asp:Label></td>
            <td>
               <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                  <ContentTemplate>
                     <table class="style1" bgcolor="#66FFFF">
                        <tr>
                           <td bgcolor="#99FFCC">
```

```
<asp:Label ID="Label2" runat="server" Font-Bold="True" Font Italic="True"  Font-
Names="Algerian" Font-Size="XX-Large" ForeColor="#000066"
Text="Label"></asp:Label>
 <br /> <asp:Timer ID="Timer1" runat="server" Interval="5000"
ontick="Timer1_Tick"> </asp:Timer>
</td>
</table>
</ContentTemplate>
</asp:UpdatePanel>
      </table>
   </div>
   </form>
</body>
</html>
```

Timer.aspx.cs code

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Timer : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      Label1.Text = "Page Load Time : " + DateTime.Now.ToLongTimeString();
   }
   protected void Timer1_Tick(object sender, EventArgs e)
   {
      Label2.Text = "Current Time : " + DateTime.Now.ToLongTimeString();
   }
}
```

15.4 WEB SERVICES

Web Services are an integral part of the .NET framework that provide a cross-platform solution for exchanging data between distributed systems. Although Web Services are normally used to allow different operating systems, object models and programming languages to send and receive data, they can also be used to dynamically inject data into an ASP.NET AJAX page or send data from a page to a back-end system. All of this can be done without resorting to postback operations.

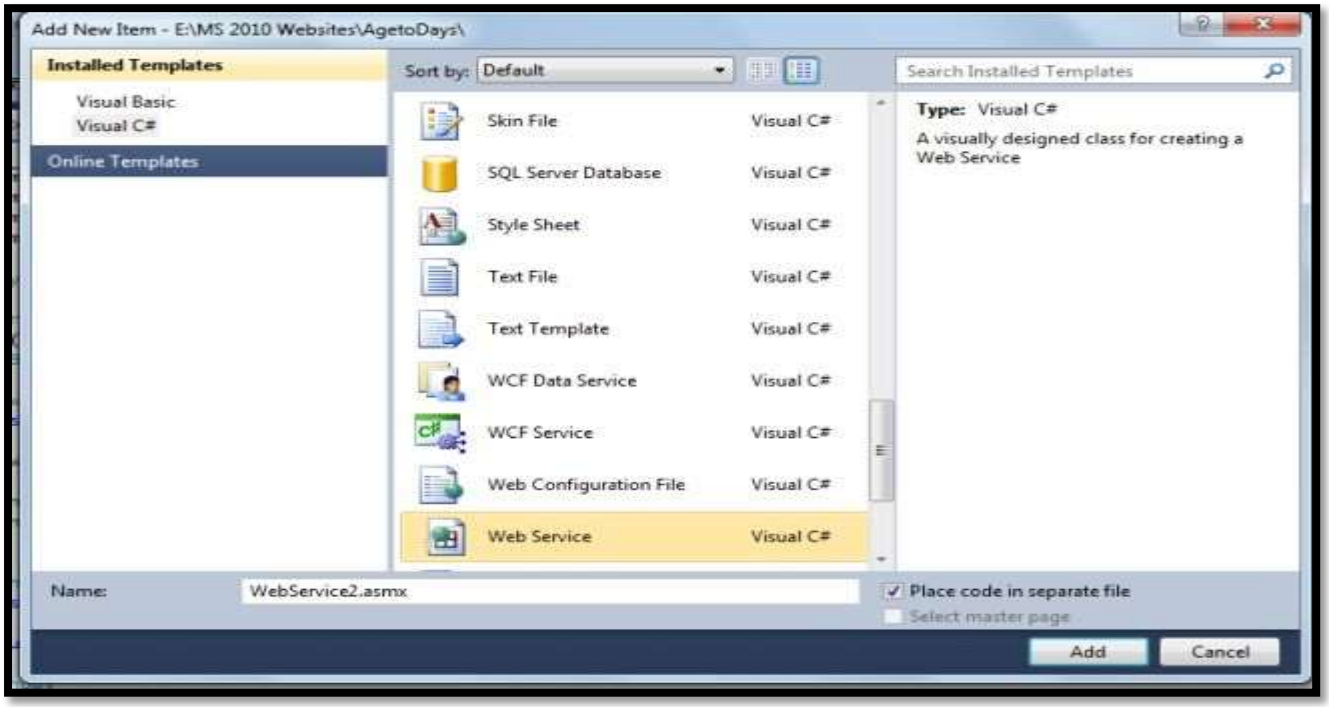
A Web Service is an application that is designed to interact directly with other applications over the internet. A WebService is the same as the Web application that can be accessed over the internet such as the Internet, and executed on a remote system hosting from requested services. The Web services are components on a Web server that a client application can call by making HTTP requests across the Web.

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications. There are three aspects of web service development:

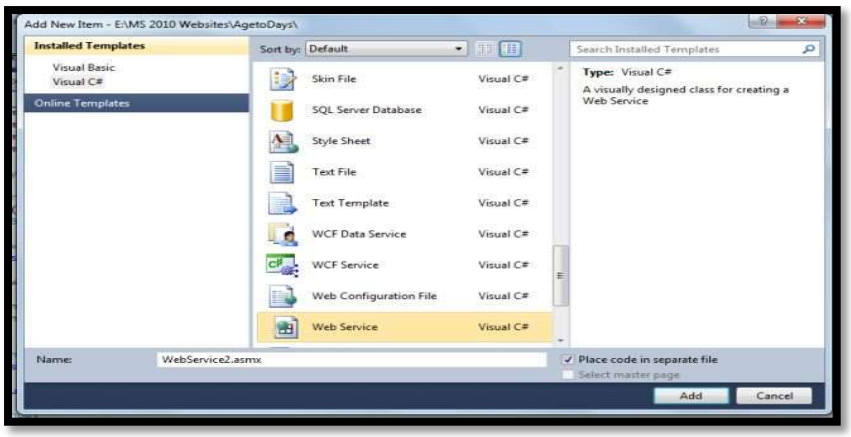
- Creating the web service
- Creating a proxy
- Consuming the web service

So let us start using a different way to add a web service using a template

1. "Start" - "All Programs" - "Microsoft Visual Studio 2010"
2. "File" - "New Project" - "C#" - "Empty Web Application" (to avoid adding a master page)
3. Provide the web site a name such as "agetodays" or another as you wish and specify the location
4. Then right-click on Solution Explorer - "Add New Item" - you see the web service templates



Select Web Service Template and click on add button. then after that the Solution Explorer look like as follows.



Then open the Webservice.cs class and write the following method followed by [webMethod] attribute as in.

```
 [WebMethod]
public int converttodaysweb(int day, int month, int year)
{
DateTime dt = new DateTime(year, month, day);
int datetodays = DateTime.Now.Subtract(dt).Days;
return datetodays;
}
```

In the code above I have declared a one integer method named converttodaysweb with the three parameters day, month and year for accepting day, month and year from the user.

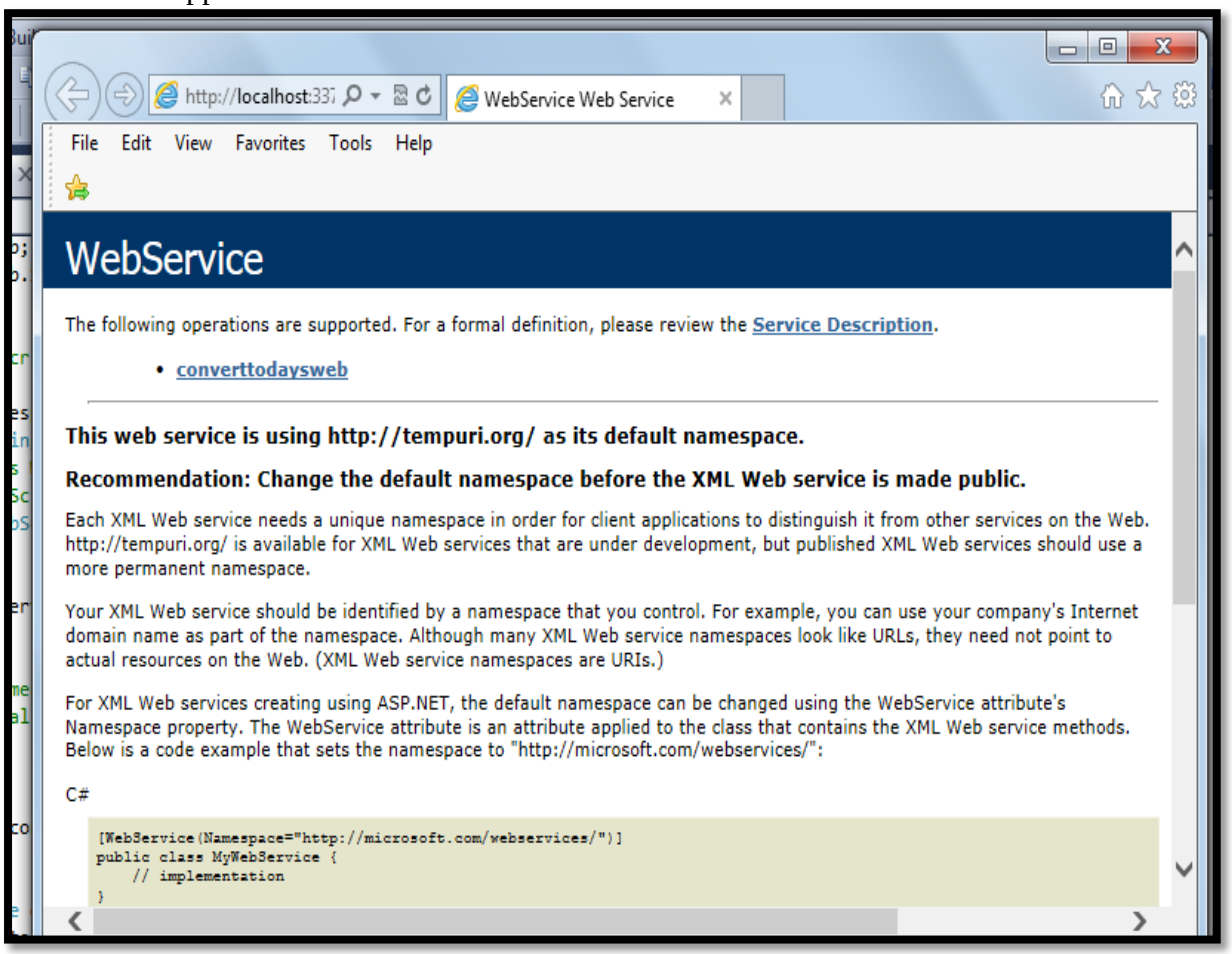
Then after that I created an object of date time and ed the those variables that I get from the users. I declared another variable in the method that is age today to store the number of days remaining from the user's input date to the current date and finally I return that variable..

The webservice.cs file will then look as in the following

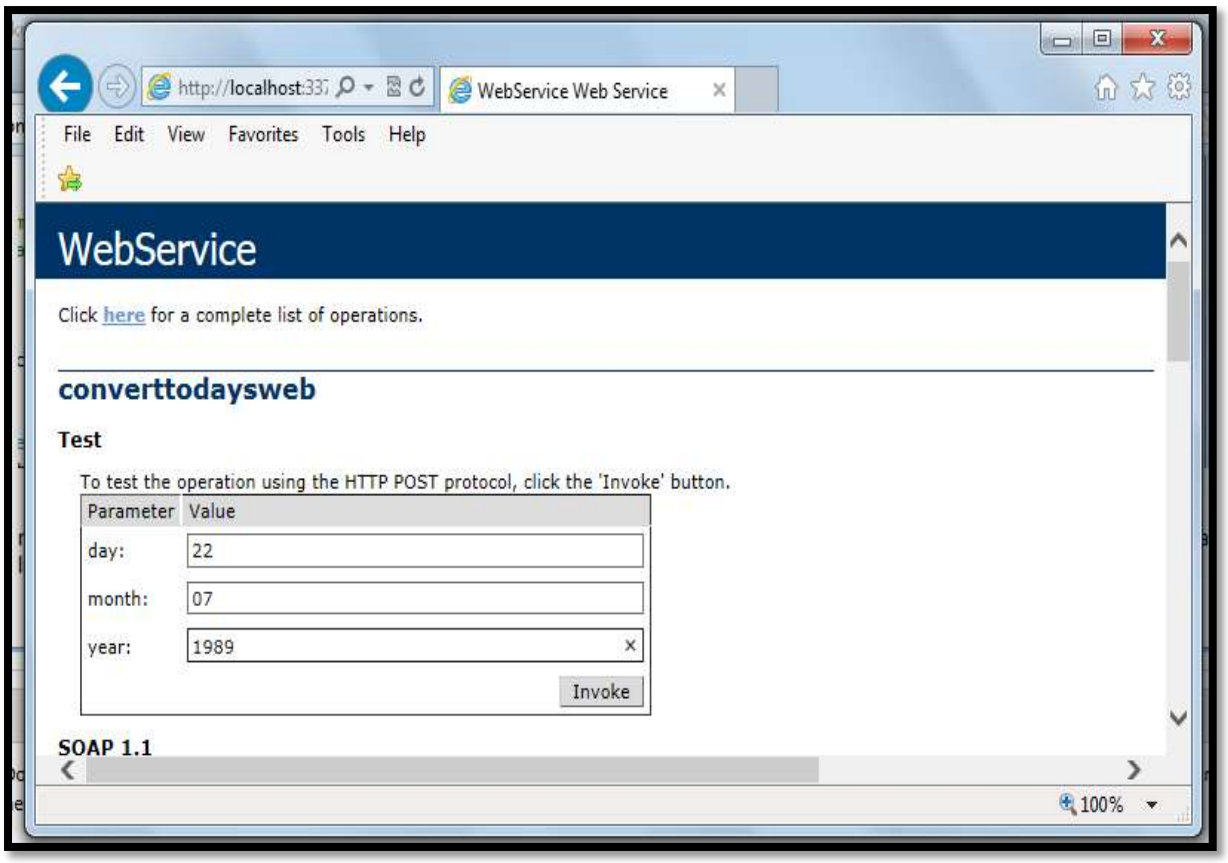
```
   using System;
   using System.Collections.Generic;
   using System.Web;
   using System.Web.Services;
   ///<summary>
   /// Summary description for UtilityWebService
   ///</summary>
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
   // [System.Web.Script.Services.ScriptService]
   public class WebService: System.Web.Services.WebService
   {
      public WebService()
      {
         //Uncomment the following line if using designed components
         //InitializeComponent();
```

```
    }
    [WebMethod]
    public int converttodaysweb(int day, int month, int year)
    {
        DateTime dt = new DateTime(year, month, day);
        int datetodays = DateTime.Now.Subtract(dt).Days;
        return datetodays;
    }
}
```

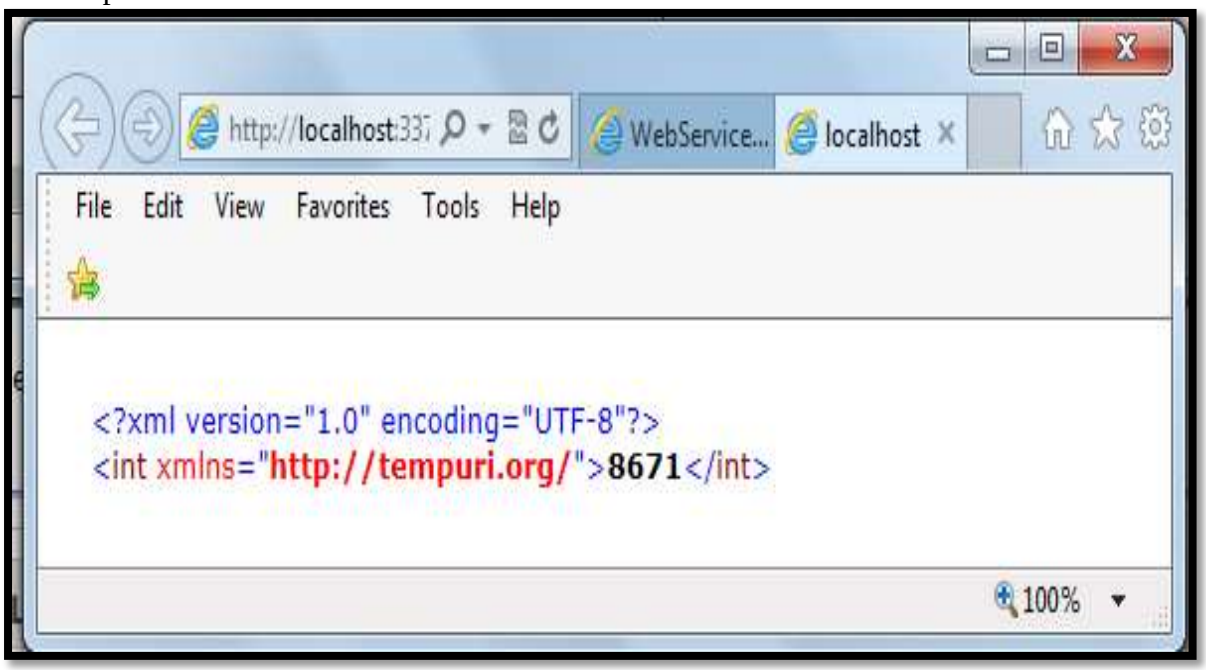
Now run the application that look like as follows.



Now in the above we see our method that we are created in the webservice.cs file, so click on that method and provide input values and click on the "invoke" link as in.



The output will be as follows



15.5 SUMMARY

This unit gives an overview of AJAX control in ASP.NET. Further it discusses about web service in ASP.NET.

15.6 EXERCISE

1. Explain the use of UpdateProgress Control and Timer Control in AJAX.
2. Explain UpdatePanel control with example.
3. Explain the use of UpdateProgress control in AJAX.
4. What is AJAX? How is the processing of a web page without AJAX different from the processing of a web page with AJAX?
5. What is AJAX? Explain UpdatePanel control with example.
6. Write a note on web service in ASP.NET.
7. What is the use of Script Manager control in AJAX.

REFERENCE

1. Beginning ASP.NET 4.5 in C# by Matthew MacDonald
2. Programming ASP.NET AJAX by Christian Wenz
3. Foundations Of Asp.net Ajax by Robin Pars Laurence Moroney John Grieb
4. Beginning Ajax with ASP.NET by Wallace B. Mcclure, Paul Glavich, Scott Cate, Craig Shoemaker
5. https://www.codeproject.com/Articles/401903/AJAX-for-Beginners-Part-Understanding-ASP-NET-AJ
6. http://ajax.net-tutorials.com/

JQuery

Content

16.0 AN INTRODUCTION TO JQUERY

- Like the ASP.NET AJAX client-side framework**,** *jQuery* is a JavaScript library that provides support for AJAX.
- In addition, jQuery contains functions that make it easier to modify documents, handle events, and apply effects and animations.
- In 2008, Microsoft adopted jQuery as part of its official application development platform and announced that it would provide official support for jQuery.
- jQuery is a lightweight open source JavaScript library (only 15kb in size) that in a relatively short span of time has become one of the most popular libraries on the web.
- A big part of the appeal of jQuery is that it allows you to elegantly (and efficiently) find and manipulate HTML elements with minimum lines of code. jQuery supports this via a nice "selector" API that allows developers to query for HTML elements, and then apply "commands" to them.

- Microsoft has included jQuery as part of ASP.NET 4, and it has included IntelliSense support for jQuery in Visual Studio 2010. In addition, Microsoft has contributed code to the jQuery project.

16.1 THE CORE JQUERY LIBRARY

- To start, jQuery makes it easy to write client-side JavaScript code that's compatible with all modern web browsers.
- In addition, it makes it easy to select and manipulate the DOM (*Document Object Model*) and the CSS for the DOM. This allows you to use JavaScript to change the appearance of the web page.
- jQuery takes this one step further by including some popular animations and effects such as having a control fade in or fade out.
- In other words, jQuery can do a lot on the client side without ever needing to make a trip to the server. However, jQuery also includes AJAX capabilities that allow it to send an AJAX request to the server and to process the AJAX response that's returned from the server.
- If you start a web site from the ASP.NET Web Site template, you will find three versions of the jQuery library in the Scripts folder.
- The jquery-1.4.1.js file contains the core library in a format that's easy for developers to read. The jquery-1.4.1.min.js file contains the core library in a condensed format that should be used when the application is deployed. And the jquery-1.4.1 -vsdoc.js file contains the core library with comments that support Visual Studio IntelliSense.

1. If your computer is not always connected to the Internet and /or if your Internet connection is not fast, you can *download jQuery* to any folder and reference it locally.
2. If your computer is always connected to the Internet, you can reference the jQuery library, indicating a Web address.

Download jQuery locally:

1. Go to the jQuery Official site and Click on "*Download jQuery*" Button in the Home page.
2. Choose: "Download the compressed, production jQuery 1.9.1", a .txt file that contains the required code will open. Copy and paste it into bloc note and save the file with "jquery.js' as name in the same folder of the Html page.
3. Copy and paste the code below between <head> and </head> :

The jQuery Ul library

- One additional feature of jQuery is its extensibility. This means that developers can develop plug-ins that is built on top of jQuery.
- Some of the most popular jQuery plug-ins can be found in the *jQuery UI* library. This library provides a wide range of user-interface controls, also known as *widgets.*
- Many of the controls in the jQuery UI library duplicate functionality that's provided by the controls in the ASP.NET AJAX Control Toolkit.
- For example, jQuery UI contains a Calendar control that's similar in function to the Calendar control in the ASP.NET AJAX Control Toolkit. As a result, if you want to use a Calendar control, you'll need to decide which control to use. The use of each one has its pros and cons.

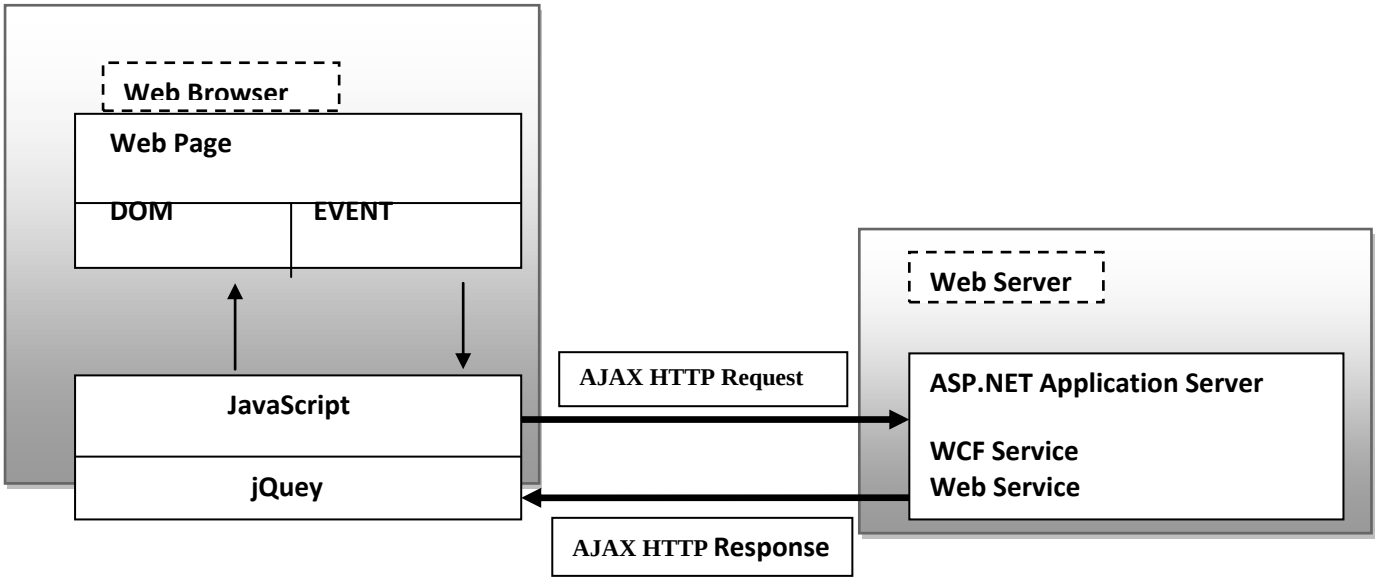
16.2 FEATURES OF JQUERY

1. **Cross-browser compatibility:** Makes it easy to write code that is compatible with all modern web browsers.
2. **Event handling:** Makes it easy to register functions as event listeners.
3. **DOM selection:** Makes it easy to select DOM elements.
4. **DOM manipulation:** Makes it easy to modify DOM elements.
5. **CSS manipulation:** Makes it easy to modify the CSS for a DOM element.
6. **Effects and animations:** Makes it easy to apply special effects and animations to DOM elements such as fading in or out, sliding up or down, and so on.
7. **AJAX:** Makes it easy to send an AJAX request to the server and use the data in the AJAX response to update the DOM for a web page.
8. **Extensibility:** Allows jQuery to work with plug-ins such as the controls in the jQuery UI library.

16.3 HOW JQUERY AJAX WORKS

- jQuery AJAX works similarly to ASP.NET AJAX. As you would expect, however, jQuery AJAX uses the jQuery library instead of the ASP.NET AJAX client-side framework.
- Then, the JavaScript code uses jQuery AJAX to call a WCF (windows communication foundation) service or web service that's running on the server.
- When a response is received from the server, the JavaScript code in the response uses jQuery to process the data and update the DOM accordingly.
- The advantage of that approach is that the JavaScript code that works with the ASP.NET AJAX client-side framework is generated automatically when the page is rendered.
- This allows the developer to quickly develop web pages that use AJAX by dragging and dropping server controls onto the page, and it shields the developer from having to understand the details of what's going on under the hood.
- The disadvantage of this approach is that the developer gives up control over how the JavaScript code works.
- The advantage of using jQuery AJAX is that the developer has more control over how the client-side controls and code work. The disadvantage of this approach is that it has a steeper learning curve. To start, the developer must have a solid understanding of HTML and CSS.
- Then, the developer must learn how to write JavaScript code that uses jQuery to work with the DOM and AJAX. Finally, the developer must manually write most of the client-side code.

16.4 THE ARCHITECTURE OF JQUERY AJAX



16.5 JQUERY SYNTAX

The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).

Basic syntax is: $(selector).action()

- A $ sign to define/access jQuery
- A (selector) to "query (or find)" HTML elements
- A jQuery action() to be performed on the element(s)

Examples:

1. $(this).hide() - hides the current element.
2. $("p").hide() - hides all <p> elements.
3. $(".test").hide() - hides all elements with class="test".
4. $("#test").hide() - hides the element with id="test".

16.6 JQUERY SELECTORS

jQuery Selectors are used to select and manipulate HTML elements. They are very important part of jQuery library. With jQuery selectors, you can find or select HTML elements based on their id, classes, attributes, types and much more from a DOM.

Selectors are used to select one or more HTML elements using jQuery and once the element is selected then you can perform various operation on that. All jQuery selectors start with a dollor sign and parenthesis e.g. $(). It is known as the factory function.

The $() factory function

Every jQuery selector start with thiis sign $(). This sign is known as the factory function. It uses the three basic building blocks while selecting an element in a given document.

S.No.	Selector	Description
1)	Tag Name:	It represents a tag name available in the DOM. For example: $('p') selects all paragraphs'p'in the document.
2)	Tag ID:	It represents a tag available with a specific ID in the DOM. For example: $('#real-id') selects a specific element in the document that has an ID of real-id.
3)	Tag Class:	It represents a tag available with a specific class in the DOM. For example: $('real-class') selects all elements in the document that have a class of real-class.

Important types of selectors in jQuery

The Universal Selector

The Universal selector, indicated by an asterisk **(*)**, applies to all elements in your page. The Universal selector can be used to set global settings like a font family. The following rule set changes the font for all elements in your page to Arial:

$(‘*’).css(‘font-family’,’Arial’);

The Type Selector

The Type selector enables you to point to an HTML element of a specific type. With a Type selector, all HTML elements of that type will be styled accordingly.

The ID Selector:

The ID selector is always prefixed by a hash symbol (#) and enables you to refer to a single element in the page. Within an HTML or ASPX page, you can give an element a unique ID using the id attribute. With the ID selector, you can change the behavior for that single element, like this:

$(‘#Button1’).addClass(‘NewClassName’);

The Class Selector:

The Class selector enables you to style multiple HTML elements through the class attribute. This is handy when you want to give the same type of formatting to a number of unrelated HTML elements. The following rule changes the text to red and bold for all HTML elements that have their class attributes set to Highlight :

<h1 class = ‘highlight’> Heading 1 </h1>
<h2> Heading 2 </h2>
** First Paragraph **
** Second Paragraph **
$(‘.highlight’).css(‘background-color’,’red’);

Grouping and Combining Selectors:

JQUERY also enables you to *group* multiple selectors by separating them with a comma. This is handy if you want to apply the same styles to different elements. The following rule turns all headings in the page to red:

$('h1, h2').css('color', 'orange');
$('#Maincontent p').css('border', '1px solid red');

16.7 JQUERY DOM

- JQuery provides methods to manipulate DOM in efficient way. You do not need to write big code to modify the value of any element's attribute or to extract HTML code from a paragraph or division.
- JQuery provides methods such as .attr(), .html(), and .val() which act as getters, retrieving information from DOM elements for later use.

  Syntax:
  selector.MethodName([parameter(s)])

  Example:
  Content Manipulation
  The html( ) method gets the html contents (innerHTML) of the first matched element.
  Here is the syntax for the method −
  Example: selector.html( )

16.8 DOM MANIPULATION METHODS

Following table lists down all the methods which you can use to manipulate DOM elements –

- after(content): Insert content after each of the matched elements.
- append(content): Append content to the inside of every matched element.
- before(content): Insert content before each of the matched elements.
- clone(bool): Clone matched DOM Elements, and all their event handlers, and select the clones.
- html(val): Set the html contents of every matched element.
- text( ) : Get the combined text contents of all matched elements.
- wrap(elem): Wrap each matched element with the specified element.

16.9 JQUERY EVENT

We have the ability to create dynamic web pages by using events. Events are actions that can be detected by your Web Application.

Following are the examples events −

- A mouse click
- A web page loading
- Taking mouse over an element

- Submitting an HTML form
- A keystroke on your keyboard, etc.

When these events are triggered, you can then use a custom function to do pretty much whatever you want with the event. These custom functions call Event Handlers.

Binding Event Handlers

Using the jQuery Event Model, we can establish event handlers on DOM elements with the **bind()** method as follows −

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $('div').bind('click', function( event ){
          alert('Hi there!');
        });
      });
    </script>

    <style>
      .div{ margin:10px;padding:12px; border:2px solid #666; width:60px;}
    </style>
  </head>

  <body>
    <p>Click on any square below to see the result:</p>

    <div class = "div" style = "background-color:blue;">ONE</div>
    <div class = "div" style = "background-color:green;">TWO</div>
    <div class = "div" style = "background-color:red;">THREE</div>
  </body>
</html>
```

The following table lists all the jQuery methods used to handle events.

bind() Attaches event handlers to elements

blur() Attaches/Triggers the blur event

change() Attaches/Triggers the change event

click() Attaches/Triggers the click event

dblclick() Attaches/Triggers the double click event

focus() Attaches/Triggers the focus event

hover() Attaches two event handlers to the hover event

keydown() Attaches/Triggers the keydown event

keypress() Attaches/Triggers the keypress event

keyup() Attaches/Triggers the keyup event

mousemove() Attaches/Triggers the mousemove event

mouseout() Attaches/Triggers the mouseout event

mouseover() Attaches/Triggers the mouseover event

mouseup() Attaches/Triggers the mouseup event

ready() Specifies a function to execute when the DOM is fully loaded

unload() Deprecated in version 1.8. Attaches an event handler to the unload event

16.10 EFFECTS WITH JQUERY

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration. This tutorial covers all the important jQuery methods to create visual effects.

Showing and Hiding Elements
The commands for showing and hiding elements are pretty much what we would expect − **show()** to show the elements in a wrapped set and **hide()** to hide them.

Syntax
Here is the simple syntax for **show()** method −
[selector].show(speed, [callback]);

Example
```
$("#hide").click(function(){
   $("p").hide();
});

$("#show").click(function(){
   $("p").show();
});
```

Syntax

$(*selector*).hide(*speed,callback*);

$(*selector*).show(*speed,callback*);
The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the hide() or show() method completes

Example
```
$("button").click(function(){
   $("p").hide(1000);
})
```

Toggling the Elements
jQuery provides methods to toggle the display state of elements between revealed or hidden. If the element is initially displayed, it will be hidden; if hidden, it will be shown.

Syntax
Here is the simple syntax for one of the **toggle()** methods −
[selector]..toggle([speed][, callback]);

Example
```
<script type = "text/javascript" language = "javascript">
        $(document).ready(function() {
          $(".clickme").click(function(event){
            $(".target").toggle('slow', function(){
              $(".log").text('Transition Complete');
            });
          });
        });
      </script>
```

jQuery Sliding Methods
With jQuery you can create a sliding effect on elements.
jQuery has the following slide methods:
- slideDown()
- slideUp()
- slideToggle()

jQuery slideDown() Method
The jQuery slideDown() method is used to slide down an element.

Syntax:
$(*selector*).slideDown(*speed,callback*);
The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the sliding completes.
The following example demonstrates the slideDown() method:

Example

```
$("#flip").click(function(){
   $("#panel").slideDown();
});
```

jQuery slideUp() Method
The jQuery slideUp() method is used to slide up an element.

Syntax:
$(*selector*).slideUp(*speed,callback*);
The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the sliding completes.
The following example demonstrates the slideUp() method:

Example

```
$("#flip").click(function(){
   $("#panel").slideUp();
});
```

jQuery slideToggle() Method
The jQuery slideToggle() method toggles between the slideDown() and slideUp() methods.
If the elements have been slid down, slideToggle() will slide them up.
If the elements have been slid up, slideToggle() will slide them down.
$(*selector*).slideToggle(*speed,callback*);
The optional speed parameter can take the following values: "slow", "fast", milliseconds.
The optional callback parameter is a function to be executed after the sliding completes.
The following example demonstrates the slideToggle() method:
Example

```
$("#flip").click(function(){
   $("#panel").slideToggle();
});
```

jQuery Fading Methods

With jQuery you can fade an element in and out of visibility.
jQuery has the following fade methods:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

jQuery fadeIn() Method
The jQuery fadeIn() method is used to fade in a hidden element.

Syntax:
$(*selector*).fadeIn(*speed,callback*);
The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the fading completes.
The following example demonstrates the fadeIn() method with different parameters:

Example

```
$("button").click(function(){
   $("#div1").fadeIn();
   $("#div2").fadeIn("slow");
   $("#div3").fadeIn(3000);
});
```

jQuery fadeOut() Method

The jQuery fadeOut() method is used to fade out a visible element.

Syntax:

$(*selector*).fadeOut(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeOut() method with different parameters:

Example

```
$("button").click(function(){
   $("#div1").fadeOut();
   $("#div2").fadeOut("slow");
   $("#div3").fadeOut(3000);
});
```

jQuery fadeToggle() Method

The jQuery fadeToggle() method toggles between the fadeIn() and fadeOut() methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

Syntax:

$(*selector*).fadeToggle(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeToggle() method with different parameters:

Example

```
$("button").click(function(){
   $("#div1").fadeToggle();
   $("#div2").fadeToggle("slow");
   $("#div3").fadeToggle(3000);
});
```

jQuery fadeTo() Method

The jQuery fadeTo() method allows fading to a given opacity (value between 0 and 1).

Syntax:

$(*selector*).fadeTo(*speed,opacity,callback*);
The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
The required opacity parameter in the fadeTo() method specifies fading to a given opacity (value between 0 and 1).
The optional callback parameter is a function to be executed after the function completes.
The following example demonstrates the fadeTo() method with different parameters:

Example

```
$("button").click(function(){
   $("#div1").fadeTo("slow", 0.15);
   $("#div2").fadeTo("slow", 0.4);
   $("# div3").fadeTo("slow", 0.7);
});
```

jQuery Animations - The animate() Method

The jQuery animate() method is used to create custom animations.

Syntax:

$(*selector*).animate({*params*},*speed,callback*);

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

Example

```
$("button").click(function(){
   $("div").animate({left: '250px'});
});
```

jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

Example with Callback

```
$("button").click(function(){
   $("p").hide("slow", function(){
      alert("The paragraph is now hidden");
   });
});
```

The example below has no callback parameter, and the alert box will be displayed before the hide effect is completed:

Example without Callback

```
$("button").click(function(){
   $("p").hide(1000);
   alert("The paragraph is now hidden");
});
```

16.11 JQUERY EXTENSIBILITY

jQuery is an open source, cross browser JavaScript library that simplifies event handling, animations and developing Ajax - enabled web pages and promotes rapid application development. The jQuery official web site states, "jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript."

jQuery is a fast, lightweight JavaScript library that is CSS3 compliant and supports many browsers. The jQuery framework is extensible and handles the DOM manipulations, CSS, AJAX, Events and Animations, very nicely.

16.12 SUMMARY

This unit gives an overview of jQuery in ASP.NET. Further it discusses about DOM, event and effect of jQuery in ASP.NET.

16.13 EXERCISE

1. What are the different types of selectors present in JQuery? Explain.
2. What is jQuery selector? Write some examples.
3. Use of document ready event and callback function of jquery.
4. Write a program using jQuery that hides a paragraph on click of a button.
5. What is the use of Document.Ready function?

6. Explain the jQuery syntax and document.Ready event with example.
7. What is jQuery?How to use jQuery.
8. Explain JQuery expression with example.
9. Write jQuery program that changes the background colour of a paragraph to red and font colour to yellow when mouse enters over it. Also set the background colour to white and font colour to black when mouse leaves the paragraph.
10. Explain the need of What is jQuery?How to use jQuery.
11. Explain DOM manipulation methods in jQuery.

REFERENCE

1. Beginning ASP.NET 4.5 in C# by Matthew MacDonald
2. Programming ASP.NET AJAX by Christian Wenz
3. Foundations Of Asp.net Ajax by Robin Pars Laurence Moroney John Grieb
4. Beginning Ajax with ASP.NET by Wallace B. Mcclure, Paul Glavich, Scott Cate, Craig Shoemaker
5. JavaScript and JQuery: Interactive Front-End Web Development by Jon Duckett
6. http://www.dotnetcurry.com/jquery/231/using-jquery-aspnet-beginner-tutorial
7. https://www.codeproject.com/Tips/471799/jQuery-introduction-and-how-to-use-jQuery-with-ASP
8. https://www.w3schools.com/Jquery/default.asp