

MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

**Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution**



DEPARTMENT OF COMPUTER SCIENCE

SUBJECT NAME: WEB TECHNOLOGY

SUBJECT CODE: SAE6B

SEMESTER: VI

PREPARED BY: PROF. S.JAMES BENEDICT FELIX

WEB TECHNOLOGY

SAE6B / SAZ6B

Unit – I

1.1 Introduction to VBScript

Microsoft VBScript (Visual Basic Script) is a general-purpose, lightweight and active scripting language developed by Microsoft that is modeled on Visual Basic. Nowadays, VBScript is the primary scripting language for Quick Test Professional (QTP), which is a test automation tool.

VBScript ("*Microsoft Visual Basic Scripting Edition*") is an Client Active Scripting language developed by Microsoft that is modeled on Visual Basic. It allows Microsoft Windows system administrators to generate powerful tools for managing computers with error handling, subroutines, and other advanced programming constructs.

Features of VBScript

- VBScript is a lightweight scripting language, which has a lightning fast interpreter.
- VBScript, for the most part, is case insensitive. It has a very simple syntax, easy to learn and to implement.
- Unlike C++ or Java, VBScript is an object-based scripting language and NOT an Object-Oriented Programming language.
- It uses Component Object Model (COM) in order to access the elements of the environment in which it is executing.
- Successful execution of VBScript can happen only if it is executed in Host Environment such as Internet Explorer (IE), Internet Information Services (IIS) and Windows Scripting Host (WSH)

VBScript – Version History and Uses

VBScript was introduced by Microsoft way back in 1996 and its first version was 1.0. The current stable version of VBScript is 5.8, which is available as part of IE8 or Windows 7. The VBScript usage areas are aplenty and not restricted to the below list.

- VBScript is used as a scripting language in one of the popular Automation testing tools – Quick Test Professional abbreviated as QTP
- Windows Scripting Host, which is used mostly by Windows System administrators for automating the Windows Desktop.
- Active Server Pages (ASP), a server side scripting environment for creating dynamic webpages which uses VBScript or Java Script.
- VBScript is used for Client side scripting in Microsoft Internet Explorer.
- Microsoft Outlook Forms usually runs on VBScript; however, the application level programming relies on VBA (Outlook 2000 onwards).

Disadvantages

- VBscript is used only by IE Browsers. Other browsers such as Chrome, Firefox DONOT Support VBScript. Hence, JavaScript is preferred over VBScript.
- VBScript has a Limited command line support.
- Since there is no development environment available by default, debugging is difficult.

Where VBScript is Today?

The current version of VBScript is 5.8, and with the recent development of .NET framework, Microsoft has decided to provide future support of VBScript within ASP.NET for web development. Hence, there will NOT be any more new versions of VBScript engine but the entire defect fixes and security issues are being addressed by the Microsoft sustaining Engineering Team. However, VBScript engine would be shipped as part of all Microsoft Windows and IIS by default.

1.2 Adding VBScript Code to an HTML Page

To insert a VBScript into an HTML page, we use the <script> tag. Inside the <script> tag, the document.write command is a standard VBScript command for writing output to a page.

There is a flexibility given to include VBScript code anywhere in an HTML document. But the most preferred way to include VBScript in your HTML file is as follows –

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

Ex:

```
<html>
<head>
<scripttype="text/Vbscript">
<!--
FunctionsayHello()
Msgbox("Hello World")
EndFunction
//-->
</script>
</head>

<body>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</body>
</html>
```

1.3 VBScript Data Types

VBScript has only one data type called a **Variant**. A **Variant** is a special kind of data type that can contain different kinds of information, depending on how it's used. Because **Variant** is the only data type in VBScript, it's also the data type returned by all functions in VBScript.

At its simplest, a **Variant** can contain either numeric or string information. A **Variant** behaves as a number when you use it in a numeric context and as a string when you use it in a string context.

The following table shows the subtypes of data that a **Variant** can contain.

Subtype	Description
Empty	Variant is uninitialized. Value is 0 for numeric variables or a zero-length string ("") for string variables.
Null	Variant intentionally contains no valid data.
Boolean	Contains either True or False .
Byte	Contains integer in the range 0 to 255.
Integer	Contains integer in the range -32,768 to 32,767.
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Long	Contains integer in the range -2,147,483,648 to 2,147,483,647.
Single	Contains a single-precision, floating-point number in the range -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
Double	Contains a double-precision, floating-point number in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for

	positive values.
Date (Time)	Contains a number that represents a date between January 1, 100 to December 31, 9999.
String	Contains a variable-length string that can be up to approximately 2 billion characters in length.
Object	Contains an object.
Error	Contains an error number.

1.4 VBScript Variables

A variable is a named memory location used to hold a value that can be changed during the script execution. VBScript has only ONE fundamental data type, Variant.

Rules for Declaring Variables –

- Variable Name must begin with an alphabet.
- Variable names cannot exceed 255 characters.
- Variables Should NOT contain a period (.)
- Variable Names should be unique in the declared context.

Declaring Variables

Variables are declared using “dim” keyword. Since there is only ONE fundamental data type, all the declared variables are variant by default. Hence, a user **NEED NOT** mention the type of data during declaration.

Example 1 – In this Example, IntValue can be used as a String, Integer or even arrays.

Dim Var

Example 2 – Two or more declarations are separated by comma(,)

Dim Variable1,Variable2

Assigning Values to the Variables

Values are assigned similar to an algebraic expression. The variable name on the left hand side followed by an equal to (=) symbol and then its value on the right hand side.

Rules

- The numeric values should be declared without double quotes.
- The String values should be enclosed within double quotes("")
- Date and Time variables should be enclosed within hash symbol(#)

Examples

' Below Example, The value 25 is assigned to the variable.

```
Value1 = 25
```

' A StringValue 'VBScript' is assigned to the variable StrValue.

```
StrValue="VBScript"
```

' The date 01/01/2020 is assigned to the variable DToday.

```
Date1 = #01/01/2020#
```

' A SpecificTimeStamp is assigned to a variable in the below example.

```
Time1=#12:30:44 PM#
```

Scope of the Variables

Variables can be declared using the following statements that determines the scope of the variable. The scope of the variable plays a crucial role when used within a procedure or classes.

- Dim
- Public
- Private

Dim

Variables declared using “Dim” keyword at a Procedure level are available only within the same procedure. Variables declared using “Dim” Keyword at script level are available to all the procedures within the same script.

1.5 VBScript Constants

Constant is a named memory location used to hold a value that CANNOT be changed during the script execution.

Declaring Constants

Syntax

```
[Public | Private] ConstConstant_Name = Value
```

The Constant can be of type Public or Private. The Use of Public or Private is Optional. The Public constants are available for all the scripts and procedures while the Private Constants are available within the procedure or Class. One can assign any value such as number, String or Date to the declared Constant.

Example 1

In this example, the value of pi is 3.4 and it displays the area of the circle in a message box.

```
<html>
<body>
<scriptlanguage="vbscript" type="text/vbscript">
DimintRadius
intRadius=20
const pi =3.14
Area= pi*intRadius*intRadius
MsgboxArea
</script>
</body>
</html>
```


Example 2

The below example illustrates how to assign a String and Date Value to a Constant.

```
<html>
<body>
<scriptlanguage="vbscript" type="text/vbscript">
ConstmyString="VBScript"
ConstmyDate=#01/01/2050#
MsgboxmyString
MsgboxmyDate
</script>
</body>
</html>
```

1.6 VBScript Operators

VBScript language supports following types of operators –

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Concatenation Operators

➤ **The Arithmetic Operators**

VBScript supports the following arithmetic operators –

Assume variable A holds 5 and variable B holds 10, then –

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give -5
*	Multiply both operands	A * B will give 50

/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B MOD A will give 0
^	Exponentiation Operator	B ^ A will give 100000

To understand these operators in a better way, you can [Try it yourself](#).

➤ The Comparison Operators

There are following comparison operators supported by VBScript language –

Assume variable A holds 10 and variable B holds 20, then –

[Show Examples](#)

Operator	Description	Example
=	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is False.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A <> B) is True.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is False.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is True.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is False.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is True.

To understand these operators in a better way, you can [Try it yourself](#).

➤ The Logical Operators

There are following logical operators supported by VBScript language –

Assume variable A holds 10 and variable B holds 0, then –

Show Examples

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True, then Expression becomes True.	$a <> 0$ AND $b <> 0$ is False.
OR	Called Logical OR Operator. If any of the two conditions is True, then condition becomes True.	$a <> 0$ OR $b <> 0$ is true.
NOT	Called Logical NOT Operator. It reverses the logical state of its operand. If a condition is True, then the Logical NOT operator will make it False.	NOT($a <> 0$ OR $b <> 0$) is false.
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to True, result is True.	$(a <> 0$ XOR $b <> 0)$ is true.

To understand these operators in a better way, you can Try it yourself.

➤ **The Concatenation Operators**

There are following Concatenation operators supported by VBScript language –

Assume variable A holds 5 and variable B holds 10 then –

Show Examples

Operator	Description	Example
+	Adds two Values as Variable Values are Numeric	A + B will give 15
&	Concatenates two Values	A & B will give 510

Assume variable A = "Microsoft" and variable B="VBScript", then –

Operator	Description	Example
+	Concatenates two Values	A + B will give MicrosoftVBScript
&	Concatenates two Values	A & B will give MicrosoftVBScript

1.7 Conditional Statements

In VBScript, there are four types of conditional statements: If...Then, If.....Then...Else, If...Then.....ElseIf, and Select Case.

In this tutorial, you will learn-

- If Then Statement
- If Else Statement
- If Elseif Statement
- SELECT Case Statement

VBScript If Then Statement

- You will use the VBScript If-Then statement if you want to execute some code when a specific condition is true.
- For example, you want to output the message "Welcome" whenever the value of the variable loggedIn is true.

```
If loggedIn = true Then
    document.write("Welcome")
End If
```

VBScript If Else Statement

- You will be using VBScript If...Then....Else statement, if you want to select one of two blocks of code to execute.

In such a case, you will be using If...Then.....Else statement.

```
If time <= 10 Then
    document.write("Hi, Good Morning")
Else
    document.write("Hi, Good Day")
```

End If

VBScript If Elseif Statement

You will be using If.....Then.....ElseIf statement, if you have to select one of many blocks of code to execute.

For example, if you want to change the output based on the day of the week, then you have to use If.....Then.....ElseIf statement.

```
If today="Sunday" Then
    document.write("Today is Sunday")
ElseIf today="Monday" Then
    document.write("Today is Monday")
ElseIf today="Tuesday" Then
    document.write("Today is Tuesday")
ElseIf today="Wednesday" Then
    document.write("Today is Wednesday")
ElseIf today="Thursday" Then
    document.write("Today is Thursday")
ElseIf today="Friday" Then
    document.write("Today is Friday")
ElseIf today="Saturday" Then
    document.write("Today is Saturday")
End If
```

VBScript SELECT Case Statement

Similar to If.....Then.....ElseIf statement, VBScript Case statement can also be used if you have to select one of many blocks of code to execute.

The same above code can be written like this using Select Case statement.

Select Case today

Case "Sunday"

```
document.write("Today is Sunday")
```

Case "Monday"

```
document.write("Today is Monday")
```

Case "Tuesday"

```
document.write("Today is Tuesday")
```

Case "Wednesday"

```
document.write("Today is Wednesday")
```

Case "Thursday"

```
document.write("Today is Thursday")
```

Case "Friday"

```
document.write("Today is Friday")
```

Case "Saturday"

```
document.write("Today is Saturday")
```

End Select

1.8 LOOPING

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in VBScript.

Loop Type	Description
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

for ..each loop	It is executed if there is at least one element in group and reiterated for each element in a group.
while..wend loop	It tests the condition before executing the loop body.
do..while loops	The do..While statements will be executed as long as condition is True.(i.e.,) The Loop should be repeated till the condition is False.
do..until loops	The do..Until statements will be executed as long as condition is False.(i.e.,) The Loop should be repeated till the condition is True.

➤ **For loop**

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

Syntax

The syntax of a **for** loop in VBScript is –

For counter = start To end [Step stepcount]

[statement 1]

[statement 2]

....

[statement n]

[Exit For]

[statement 11]

[statement 22]

....

[statement n]

Next

Example

```
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    Dim a : a = 10
    For i = 0 to a Step 2 'i is the counter variable and it is incremented by 2
```

```

document.write("The value is i is : " &i)
document.write("<br></br>")
    Next

</script>
</body>
</html>

```

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

The value is i is : 0

The value is i is : 2

The value is i is : 4

The value is i is : 6

The value is i is : 8

The value is i is : 10

➤ **For Each Loop**

A **For Each** loop is used when we want to execute a statement or a group of statements for each element in an array or collection.

Example

```

<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    'fruits is an array
    fruits = Array("apple","orange","cherries")
    Dim fruitnames
    'iterating using For each loop.
    For each item in fruits
fruitnames = fruitnames&item&vbnewline

```



```

Next
msgboxfruitnames
</script>
</body>
</html>

```

When the above code is executed, it prints all the fruitnames with one item in each line.

```

apple
orange
cherries

```

➤ **while.....wend**

In a **While..Wend** loop, if the condition is True, all statements are executed until **Wend** keyword is encountered.

If the condition is false, the loop is exited and the control jumps to very next statement after **Wend** keyword.

Example

```

<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    Dim Counter : Counter = 10
    While Counter < 15 ' Test value of Counter.
        Counter = Counter + 1 ' Increment Counter.
    document.write("The Current Value of the Counter is : " & Counter)
    document.write("<br></br>")
    Wend ' While loop exits if Counter Value becomes 15.
</script>
</body>
</html>

```

When the above code is executed, it prints the following output in the console.

The Current Value of the Counter is : 11
The Current Value of the Counter is : 12
The Current Value of the Counter is : 13
The Current Value of the Counter is : 14
The Current Value of the Counter is : 15

Do.....while Loop

A **Do..While** loop is used when we want to repeat a set of statements as long as the condition is true. The Condition may be checked at the beginning of the loop or at the end of the loop.

Example

```
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    Do While i< 5
i = i + 1
Document.write("The value of i is : " &i)
Document.write("<br></br>")
    Loop
</script>
</body>
</html>
```

When the above code is executed, it prints the following output on the console.

The value of i is : 1
The value of i is : 2
The value of i is : 3

The value of i is : 4

The value of i is : 5

Do.....Until Loop

A **Do..Until** loop is used when we want to repeat a set of statements as long as the condition is false. The Condition may be checked at the beginning of the loop or at the end of loop.

Example

```
<html>
<body>
<script language="vbscript" type="text/vbscript">
i=10
Do Untili>15'Condition is False.Hence loop will be executed
i=i+1
Document.write("The value of i is : "&i)
Document.write("<br></br>")
Loop

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

The value of i is : 11

The value of i is : 12

The value of i is : 13

The value of i is : 14

The value of i is : 15

The value of i is : 16

1.9 VBScript Procedures

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing same code over and over again. This will enable programmers to divide a big program into a number of small and manageable functions.

Example

```
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    Function sayHello()
    MsgBox("Hello there")
    End Function

</script>
</body>
</html>
```

Calling a Function

To invoke a function somewhere later in the script, you would simple need to write the name of that function with the **Call** keyword.

```
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
    Function sayHello()
    MsgBox("Hello there")
    End Function
    Call sayHello()z

</script>
</body>
</html>
```

1.10 VBScript Functions

VBScript has the following functions

- Date / Time Functions
- Conversions Functions
- Format functions
- Math Functions
- Array Functions
- String Functions
- Other Functions

Date/Time Functions

Function	Description
<u>CDate</u>	Converts a valid date and time expression to the variant of subtype Date
<u>Date</u>	Returns the current system date
<u>DateAdd</u>	Returns a date to which a specified time interval has been added
<u>DateDiff</u>	Returns the number of intervals between two dates
<u>DatePart</u>	Returns the specified part of a given date
<u>DateSerial</u>	Returns the date for a specified year, month, and day
<u>DateValue</u>	Returns a date
<u>Day</u>	Returns a number that represents the day of the month (between 1 and 31, inclusive)
<u>FormatDateTime</u>	Returns an expression formatted as a date or time
<u>Hour</u>	Returns a number that represents the hour of the day (between 0 and 23, inclusive)
<u>IsDate</u>	Returns a Boolean value that indicates if the evaluated expression can be converted to

	a date
<u>Minute</u>	Returns a number that represents the minute of the hour (between 0 and 59, inclusive)
<u>Month</u>	Returns a number that represents the month of the year (between 1 and 12, inclusive)
<u>MonthName</u>	Returns the name of a specified month
<u>Now</u>	Returns the current system date and time
<u>Second</u>	Returns a number that represents the second of the minute (between 0 and 59, inclusive)
<u>Time</u>	Returns the current system time
<u>Timer</u>	Returns the number of seconds since 12:00 AM
<u>TimeSerial</u>	Returns the time for a specific hour, minute, and second
<u>TimeValue</u>	Returns a time
<u>Weekday</u>	Returns a number that represents the day of the week (between 1 and 7, inclusive)
<u>WeekdayName</u>	Returns the weekday name of a specified day of the week
<u>Year</u>	Returns a number that represents the year

Conversion Functions

Function	Description
<u>Asc</u>	Converts the first letter in a string to ANSI code
<u>CBool</u>	Converts an expression to a variant of subtype Boolean
<u>CByte</u>	Converts an expression to a variant of subtype Byte
<u>CCur</u>	Converts an expression to a variant of subtype Currency
<u>CDate</u>	Converts a valid date and time expression to the variant of subtype Date
<u>CDbl</u>	Converts an expression to a variant of subtype Double
<u>Chr</u>	Converts the specified ANSI code to a character

<u>CInt</u>	Converts an expression to a variant of subtype Integer
<u>CLng</u>	Converts an expression to a variant of subtype Long
<u>CSng</u>	Converts an expression to a variant of subtype Single
<u>CStr</u>	Converts an expression to a variant of subtype String
<u>Hex</u>	Returns the hexadecimal value of a specified number
<u>Oct</u>	Returns the octal value of a specified number

Format Functions

Function	Description
<u>FormatCurrency</u>	Returns an expression formatted as a currency value
<u>FormatDateTime</u>	Returns an expression formatted as a date or time
<u>FormatNumber</u>	Returns an expression formatted as a number
<u>FormatPercent</u>	Returns an expression formatted as a percentage

Math Functions

Function	Description
<u>Abs</u>	Returns the absolute value of a specified number
<u>Atn</u>	Returns the arctangent of a specified number
<u>Cos</u>	Returns the cosine of a specified number (angle)
<u>Exp</u>	Returns e raised to a power
<u>Hex</u>	Returns the hexadecimal value of a specified number
<u>Int</u>	Returns the integer part of a specified number
<u>Fix</u>	Returns the integer part of a specified number
<u>Log</u>	Returns the natural logarithm of a specified number
<u>Oct</u>	Returns the octal value of a specified number

<u>Rnd</u>	Returns a random number less than 1 but greater or equal to 0
<u>Sgn</u>	Returns an integer that indicates the sign of a specified number
<u>Sin</u>	Returns the sine of a specified number (angle)
<u>Sqr</u>	Returns the square root of a specified number
<u>Tan</u>	Returns the tangent of a specified number (angle)

Array Functions

Function	Description
<u>Array</u>	Returns a variant containing an array
<u>Filter</u>	Returns a zero-based array that contains a subset of a string array based on a filter criteria
<u>IsArray</u>	Returns a Boolean value that indicates whether a specified variable is an array
<u>Join</u>	Returns a string that consists of a number of substrings in an array
<u>LBound</u>	Returns the smallest subscript for the indicated dimension of an array
<u>Split</u>	Returns a zero-based, one-dimensional array that contains a specified number of substrings
<u>UBound</u>	Returns the largest subscript for the indicated dimension of an array

String Functions

Function	Description
<u>InStr</u>	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
<u>InStrRev</u>	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
<u>LCase</u>	Converts a specified string to lowercase
<u>Left</u>	Returns a specified number of characters from the left side of a string
<u>Len</u>	Returns the number of characters in a string
<u>LTrim</u>	Removes spaces on the left side of a string

<u>RTrim</u>	Removes spaces on the right side of a string
<u>Trim</u>	Removes spaces on both the left and the right side of a string
<u>Mid</u>	Returns a specified number of characters from a string
<u>Replace</u>	Replaces a specified part of a string with another string a specified number of times
<u>Right</u>	Returns a specified number of characters from the right side of a string
<u>Space</u>	Returns a string that consists of a specified number of spaces
<u>StrComp</u>	Compares two strings and returns a value that represents the result of the comparison
<u>String</u>	Returns a string that contains a repeating character of a specified length
<u>StrReverse</u>	Reverses a string
<u>UCase</u>	Converts a specified string to uppercase

Other Functions

Function	Description
<u>CreateObject</u>	Creates an object of a specified type
<u>Eval</u>	Evaluates an expression and returns the result
<u>IsEmpty</u>	Returns a Boolean value that indicates whether a specified variable has been initialized or not
<u>IsNull</u>	Returns a Boolean value that indicates whether a specified expression contains no valid data
<u>IsNumeric</u>	Returns a Boolean value that indicates whether a specified expression can be evaluated as a number
<u>IsObject</u>	Returns a Boolean value that indicates whether the specified expression is an automationObject
<u>RGB</u>	Returns a number that represents an RGB color value

<u>Round</u>	Rounds a number
<u>ScriptEngine</u>	Returns the scripting language in use
<u>ScriptEngineBuildVersion</u>	Returns the build version number of the scripting engine in use
<u>ScriptEngineMajorVersion</u>	Returns the major version number of the scripting engine in use
<u>ScriptEngineMinorVersion</u>	Returns the minor version number of the scripting engine in use
<u>TypeName</u>	Returns the subtype of a specified variable
<u>VarType</u>	Returns a value that indicates the subtype of a specified variable

1.11 VBScript Coding Conventions

Coding conventions are suggestions that help you write code using Microsoft Visual Basic Scripting Edition.

Coding conventions can include the following:

- Naming conventions for objects, variables, and procedures
- Commenting conventions
- Text formatting and indenting guidelines

The main reason for using a consistent set of coding conventions is to standardize the structure and coding style of a script.

Using good coding conventions results in precise, readable, and unambiguous source code that is consistent with other language conventions and as intuitive as possible.

Constant Naming Conventions

Constants were implemented as variables and distinguished from other variables using all uppercase characters.

Multiple words were separated using the underscore (_) character.

For example:

`USER_LIST_MAX`

NEW_LINE

By using an alternative naming scheme, we can create true constants using the Const statement. This convention uses a mixed-case format in which constant names have a "con" prefix.

For example: conYourOwnConstant

Variable Naming Conventions:

For purposes of readability and consistency, we can use the following prefixes with descriptive names for variables in our VBScript code.

SUBTYPE	PREFIX	EXAMPLE
Boolean	bln	blnFound
Byte	byt	bytRasterData
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

1.12 Dictionary Object in VBScript

The Dictionary object is used to hold a set of data values in the form of (key, item) pairs. A dictionary is sometimes called an associative array because it associates a key with an item. The keys behave in a way similar to indices in an array, except that array indices are numeric and keys are arbitrary strings. Each key in a single Dictionary object must be unique.

Dictionaries are frequently used when some items need to be stored and recovered by name. For example, a dictionary can hold all the environment variables defined by the system or all the

values associated with a registry key. However, a dictionary can only store one item for each key value. That is, dictionary keys must all be unique.

Creating Dictionaries

To construct an instance of a dictionary object, just use the following lines of code:

```
DimobjDictionary
SetobjDictionary = CreateObject("Scripting.Dictionary")
```

1.13 Err Object

The VBScript `Err` object provides access to run- time error information.

The `Err` object encapsulates errors for a VBScript script. By default, if an error occurs, VBScript terminates script execution and RhinoScript reports the error back to the user. Sometimes this default error processing is not desirable. In this case, the `Err` object and the `On Error` statement can be used to let scripts perform their own error handling.

Using On Error

To generate a user-defined run-time error, first clear the `Err` object using the `.Clear` method. Then raise the error using the `.Raise` method. This method takes up to five arguments that correspond, in order, to the properties previously listed. For example:

```
Err.Clear
Err.Raise 1000, "This is a script-defined error", "Test Script"
```

This example displays the standard RhinoScript error dialog box showing the error information.

To intercept run-time errors and process them in scripts, use the `On Error` statement. The syntax of this statement is:

```
OnErrorResumeNext
```

Unit – II

2.1 Introduction to JavaScript

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.

JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.). There are also more advanced server side versions of JavaScript such as Node.js. JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.

Features of JavaScript

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform
- JavaScript is a object-based scripting language.
- Giving the user more control over the browser.
- It Handling dates and time.
- It Detecting the user's browser and OS,
- It is light weighted.
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter based scripting language.
- JavaScript is case sensitive.
- JavaScript is object based language as it provides predefined objects.

- Every statement in JavaScript must be terminated with semicolon (;).
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.
- An important part of JavaScript is the ability to create new functions within scripts.

Declare a function in JavaScript using function keyword.

2. 2 Advantages of JavaScript

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.
- **Speed.** Client-side JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server.
- **Simplicity.** JavaScript is relatively simple to learn and implement.
- **Popularity** JavaScript is used everywhere on the web.
- **Interoperability.** JavaScript plays nicely with other languages and can be used in a huge variety of applications.

- **Server Load.** Being client-side reduces the demand on the website server.
- Gives the ability to create rich interfaces

JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
Var x, y, z;    // Declar Variables
X = 5; y = 6;  // Assign Values
Z = x + y;     // Compute Values
```

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called **Literals**.

Variable values are called **Variables**.

JavaScript Literals

The two most important syntax rules for fixed values are:

1. Numbers are written with or without decimals:
10.50
1001
2. Strings are text, written within double or single quotes:
"John Doe"
'John Doe'

2.3 Data Types in JavaScript

Data types basically specify what kind of data can be stored and manipulated within a program. There are six basic data types in JavaScript which can be divided into three main categories: primitive (or primary), composite (or reference), and special data types.

String, Number, and Boolean are primitive data types. Object, Array, and Function (which are all types of objects) are composite data types. Whereas Undefined and Null are special data types.

Primitive data types can hold only one value at a time, whereas composite data types can hold collections of values and more complex entities.

- Numbers, eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object.

Note – JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

a. JavaScript Variable

In a programming language, variables are used to store data values. JavaScript uses the var keyword to declare variables. An equal sign is used to assign values to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
Var x;
```

```
X = 6;
```

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable. There are some rules while declaring a JavaScript variable (also known as identifiers).

Syntax:

```
Var<variable-name>;
```


Var<variable-name> = <value>;

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScriptJavaScript variables are case sensitive, for example x and X are different variables.

Example of JavaScript variables

JavaScript variables are containers for storing data values. In this example, x, y, and z, are variables, declared with the var keyword:

Example

```
Var x = 5;
```

```
Var y = 6;
```

```
Var z = x + y;
```

From the example above, you can expect:

X stores the value 5

Y stores the value 6

Z stores the value 11

Global Variables – A global variable has global scope which means it can be defined anywhere in your JavaScript code.

Example

```
Var carName = "Volvo";
```

```
// code here can use carName
```

```
Function myFunction() {
```

```
// code here can also use carName
```

```
}
```

Local Variables – A local variable will be visible only within a function where it is defined.

Function parameters are always local to that function.

Example

```
// code here can NOT use carName
```

```
Function myFunction() {
```

```
  VarcarName = "Volvo";
```

```
  // code here CAN use carName
```

```
}
```

2.4 Array

The JavaScript Array class is a global object that is used in the construction of arrays; which are high-level, list-like objects.

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements. There are 3 ways to construct array in JavaScript

- By array literal
- By creating instance of Array directly (using new keyword)
- By using an Array constructor (using new keyword)

What is an Array?

An array is an object that can store a collection of items. Arrays become really useful when you need to store large amounts of data of the same type.

Suppose you want to store details of 500 employees. If you are using variables, you will have to create 500 variables whereas you can do the same with a single array. You can access the items in an array by referring to its index number and the index of the first element of an array is zero.

Syntax

Use the following syntax to create an **Array** object –

```
var fruits = new Array( "apple", "orange", "mango" );
```

JavaScript array literal

The syntax of creating array using array literal is given below:

```
Vararrayname=[value1,value2.....valueN];
```

As you can see, values are contained inside [] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

```
<script>
Varemp=["Sonoo","Vimal","Ratan"];
For (i=0;i<emp.length;i++){
Document.write(emp[i] + "<br/>");
}
</script>
```

Output of the above example

Sonoo

Vimal

Ratan

JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
Vararrayname=new Array();
```

Here, new keyword is used to create instance of array.

Let's see the example of creating array directly.

```
<script>
Var I;
Varemp = new Array();
```

```

Emp[0] = "Arun";
Emp[1] = "Varun";
Emp[2] = "John";
For (i=0;i<emp.length;i++){
Document.write(emp[i] + "<br>");
}
</script>

```

Output of the above example

Arun

Varun

John

JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

```

<script>
Varemp=new Array("Jai","Vijay","Smith");
For (i=0;i<emp.length;i++){
Document.write(emp[i] + "<br>");
}
</script>

```

Test it Now

Output of the above example

Jai

Vijay

Smith

2.5 Javascript Operators

JavaScript includes operators as in other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example 1 +

2, where + sign is an operator and 1 is left operand and 2 is right operand. + operator adds two numeric values and produces a result which is 3 in this case.

Syntax:

<Left operand> operator <right operand>

<Left operand> operator

JavaScript includes following categories of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Conditional Operator
7. typeof Operator

2.5.1 Arithmetic operator

Arithmetic operators are used to perform arithmetic calculations. For example,

```
const number = 3 + 5; // 8
```

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Remainder	$x \% y$
++	Increment (increments by 1)	++x or x++
--	Decrement (decrements by 1)	--x or x--

** Exponentiation (Power) x **

1. + (Addition)

Adds two operands

Ex: A + B will give 30

2. (Subtraction)

Subtracts the second operand from the first

Ex: A – B will give -10

3. (Multiplication)

Multiply both operands

Ex: A * B will give 200

4. / (Division)

Divide the numerator by the denominator

Ex: B / A will give 2

5. % (Modulus)

Outputs the remainder of an integer division

Ex: B % A will give 0

6. ++ (Increment)

Increases an integer value by one

Ex: A++ will give 11

7. (Decrement)

Decreases an integer value by one

Ex: A—will give 9

<html>

<body>

<script type = “text/javascript”>

```
Var a = 33;
Var b = 10;
Var c = "Test";
Varlinebreak = "<br />";
    Document.write("a + b = ");
    Result = a + b;
    Document.write(result);
    Document.write(linebreak);
    Document.write("a - b = ");
        Result = a - b;
    Document.write(result);
    Document.write(linebreak);
    Document.write("a / b = ");
        Result = a / b;
    Document.write(result);
    Document.write(linebreak);
    Document.write("a % b = ");
        Result = a % b;
    Document.write(result);
    Document.write(linebreak);
    Document.write("a + b + c = ");
        Result = a + b + c;
    Document.write(result);
    Document.write(linebreak);
```

```
A = ++a;
Document.write(++a = );
    Result = ++a;
Document.write(result);
Document.write(linebreak);
```

```

    B = --b;
    Document.write("--b = ");
    Result = --b;
    Document.write(result);
    Document.write(linebreak);

```

```

</script>
</body>
</html>

```

Output

```

A + b = 43
A - b = 23
A / b = 3.3
A % b = 3
A + b + c = 43Test
++a = 35
--b = 8

```

2.5.2 Comparison Operators

Returns true if the operands are equal and of the same type. See also Object is and sameness in JS. 3 === var1. Strict not equal (!==) Returns true if the operands are of the same type but not equal, or are of different type.

JavaScript supports the following comparison operators –

Assume variable A holds 10 and variable B holds 20, then –

| Sr.No. | Operator & Description |
|---------------|---|
| 1 | = = (Equal)Checks if the value of two operands are equal or not, if yes, then the condition becomes true. |

Ex: (A == B) is not true.

- 2 != (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.

Ex: (A != B) is true.

- 3 > (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.

Ex: (A > B) is not true.

- 4 < (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.

Ex: (A < B) is true.

- 5 >= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A >= B) is not true.

- 6 <= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A <= B) is true.

Example

```
<html>
<body>
<script type = "text/javascript">
<!--
```

```
Var a = 10;
Var b = 20;
Varlinebreak = "<br />";

Document.write("(a == b) => ");
    Result = (a == b);
Document.write(result);
Document.write(linebreak);

Document.write("(a < b) => ");
    Result = (a < b);
Document.write(result);
Document.write(linebreak);

Document.write("(a > b) => ");
    Result = (a > b);
Document.write(result);
Document.write(linebreak);

Document.write("(a != b) => ");
    Result = (a != b);
Document.write(result);
Document.write(linebreak);

Document.write("(a >= b) => ");
    Result = (a >= b);
Document.write(result);
Document.write(linebreak);

Document.write("(a <= b) => ");
    Result = (a <= b);
Document.write(result);
```

```
Document.write(linebreak);
```

```
</script>
```

Set the variables to different values and different operators and then try...

```
</body>
```

```
</html>
```

Output

```
(a == b) => false
```

```
(a < b) => true
```

```
(a > b) => false
```

```
(a != b) => true
```

```
(a >= b) => false
```

```
A <= b) => true
```

2.5.3 Logical Operators

A logical operator is a symbol or word used to connect two or more expressions such that the value of the compound expression produced depends only on that of the original expressions and on the meaning of the operator. Common logical operators include AND, OR, and NOT.

JavaScript supports the following logical operators –

Assume variable A holds 10 and variable B holds 20, then –

Sr.No.	Operator & Description
1	<p>&& (Logical AND) If both the operands are non-zero, then the condition becomes true.</p> <p>Ex: (A && B) is true.</p>
2	<p> (Logical OR)</p> <p>If any of the two operands are non-zero, then the condition becomes true.</p>

Ex: (A || B) is true.

3 ! (Logical NOT)

Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Ex: !(A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

Live Demo

```
<html>
<body>
<script type = "text/javascript">
    Var a = true;
    Var b = false;
    Varlinebreak = "<br />";

    Document.write("(a && b) => ");
    Result = (a && b);
    Document.write(result);
    Document.write(linebreak);

    Document.write("(a || b) => ");
    Result = (a || b);
    Document.write(result);
    Document.write(linebreak);
```

```

Document.write("!(a && b) => ");
Result = (!(a && b));
Document.write(result);
Document.write(linebreak);

```

```

</script>
</body>
</html>

```

Output

(a && b) => false

(a || b) => true

!(a && b) => true

Set the variables to different values and different operators and then try...

2.5.4 Bitwise Operators

JavaScript stores numbers as 64 bits floating point numbers, but all bitwise operations are performed on 32 bits binary numbers. Before a bitwise operation is performed, JavaScript converts numbers to 32 bits signed integers. ... A signed integer uses the leftmost bit as the minus sign.(-)

JavaScript supports the following bitwise operators –

Assume variable A holds 2 and variable B holds 3, then –

Sr.No.	Operator & Description
1	<p>& (Bitwise AND)</p> <p>It performs a Boolean AND operation on each bit of its integer arguments. Ex: (A & B) is 2.</p>
2	<p> (BitWise OR)</p> <p>It performs a Boolean OR operation on each bit of its integer arguments.</p>

	Ex: (A B) is 3.
3	<p>^ (Bitwise XOR)</p> <p>It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.</p> <p>Ex: (A ^ B) is 1.</p>
4	<p>~ (Bitwise Not)</p> <p>It is a unary operator and operates by reversing all the bits in the operand.</p> <p>Ex: (~B) is -4.</p>
5	<p><< (Left Shift)</p> <p>It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.</p> <p>Ex: (A << 1) is 4.</p>
6	<p>>> (Right Shift)</p> <p>Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.</p> <p>Ex: (A >> 1) is 1.</p>
7	<p>>>> (Right shift with Zero)</p> <p>This operator is just like the >> operator, except that the bits shifted in on the left are always zero.</p> <p>Ex: (A >>> 1) is 1.</p>

Example

```

<html>
<body>
<script type = "text/javascript">
    Var a = 2; // Bit presentation 10
    Var b = 3; // Bit presentation 11
    Varlinebreak = "<br />";

```

```
Document.write("(a & b) => ");
Result = (a & b);
Document.write(result);
Document.write(linebreak);
Document.write("(a | b) => ");
    Result = (a | b);
Document.write(result);
Document.write(linebreak);
Document.write("(a ^ b) => ");
    Result = (a ^ b);
Document.write(result);
Document.write(linebreak);
Document.write("(~b) => ");
    Result = (~b);
Document.write(result);
Document.write(linebreak);
Document.write("(a << b) => ");
    Result = (a << b);
Document.write(result);
Document.write(linebreak);
Document.write("(a >> b) => ");
    Result = (a >> b);
Document.write(result);
Document.write(linebreak);
```

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

OUTPUT

(a & b) => 2

(a | b) => 3

(a ^ b) => 1

(~b) => -4

(a << b) => 16

(a >> b) => 0

Set the variables to different values and different operators and then try...

2.5.5 Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand. That is, $a = b$ assigns the value of b to a In addition to the regular assignment operator “=”

JavaScript supports the following assignment operators –

S.NO	Operator & Description
1	<p>= (Simple Assignment)</p> <p>Assigns values from the right side operand to the left side operand</p> <p>Ex: $C = A + B$ will assign the value of $A + B$ into C</p>
2	<p>+= (Add and Assignment)</p> <p>It adds the right operand to the left operand and assigns the result to the left operand.</p> <p>Ex: $C += A$ is equivalent to $C = C + A$</p>
3	<p>-= (Subtract and Assignment)</p> <p>It subtracts the right operand from the left operand and assigns the result to the left operand.</p> <p>Ex: $C -= A$ is equivalent to $C = C - A$</p>
4	<p>*= (Multiply and Assignment)</p>

	<p>It multiplies the right operand with the left operand and assigns the result to the left operand.</p> <p>Ex: $C *= A$ is equivalent to $C = C * A$</p>
5	<p>$/=$ (Divide and Assignment)</p> <p>It divides the left operand with the right operand and assigns the result to the left operand.</p> <p>Ex: $C /= A$ is equivalent to $C = C / A$</p>
6	<p>$\%=$ (Modules and Assignment)</p> <p>It takes modulus using two operands and assigns the result to the left operand.</p> <p>Ex: $C \% = A$ is equivalent to $C = C \% A$</p> <p>Note – Same logic applies to Bitwise operators so they will become like $\ll=$, $\gg=$, $\>>=$, $\&=$, $=$ and $\^=$.</p>

Example

```
<html>
<body>
<script type = "text/javascript">
<!--
Var a = 33;
Var b = 10;
Varlinebreak = "<br />";
```

```
Document.write("Value of a => (a = b) => ");
```

```
Result = (a = b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
Document.write("Value of a => (a += b) => ");
```

```
Result = (a += b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
Document.write("Value of a => (a -= b) => ");
```

```
    Result = (a -= b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
Document.write("Value of a => (a *= b) => ");
```

```
    Result = (a *= b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
Document.write("Value of a => (a /= b) => ");
```

```
    Result = (a /= b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
Document.write("Value of a => (a %= b) => ");
```

```
    Result = (a %= b);
```

```
Document.write(result);
```

```
Document.write(linebreak);
```

```
</script>
```

```
<p>Set the variables to different values and different operators and then try...</p>
```

```
</body>
```

```
</html>
```

Output

Value of a => (a = b) => 10

Value of a => (a += b) => 20

Value of a => (a -= b) => 10

Value of a => (a *= b) => 100

Value of a => (a /= b) => 10

Value of $a \Rightarrow (a \% b) \Rightarrow 0$

2.5.6 Conditional Operator

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No.	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

```
<html>
<body>
<script type = "text/javascript">
<!--
Var a = 10;
Var b = 20;
Varlinebreak = "<br />";

    Document.write ("((a > b) ? 100 : 200) => ");
        Result = (a > b) ? 100 : 200;
    Document.write(result);
    Document.write(linebreak);

    Document.write ("((a < b) ? 100 : 200) => ");
        Result = (a < b) ? 100 : 200;
    Document.write(result);
    Document.write(linebreak);
```

```
</script>
```

```
</body>
```

```
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
```

```
((a < b) ? 100 : 200) => 100
```

2.5.7 Typeof Operator

The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The typeof operator evaluates to “number”, “string”, or “boolean” if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the type of Operator.

Type	String Returned by typeof
Number	“number”
String	“string”
Boolean	“boolean”
Object	“object”
Function	“function”
Undefined	“undefined”
Null	“object”

Example

```
<html>
```

```
<body>
```

```
<script type = “text/javascript”>
```

```

<!--
Var a = 10;
Var b = "String";
Varlinebreak = "<br />";

    Result = (typeof b == "string" ? "B is String" : "B is Numeric");
Document.write("Result => ");
Document.write(result);
Document.write(linebreak);

    Result = (typeof a == "string" ? "A is String" : "A is Numeric");
Document.write("Result => ");
Document.write(result);
Document.write(linebreak);

</script>
</body>
</html>

```

Output

```

Result => B is String
Result => A is Numeric

```

2.6 JavaScript Expressions

Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. As per the MDN documentation, JavaScript has the following expression categories.

- **Arithmetic Expressions:**

Arithmetic expressions evaluate to a numeric value. Examples include the following

```
10; // Here 10 is an expression that is evaluated to the numeric value 10 by the JS interpreter
```

```
10+13; // This is another expression that is evaluated to produce the numeric value 23
```

- **String Expressions:**

String expressions are expressions that evaluate to a string. Examples include the following

`'hello';`

`'hello' + 'world';` // evaluates to the string `'hello world'`

- **Logical Expressions:**

Expressions that evaluate to the boolean value true or false are considered to be logical expressions. This set of expressions often involve the usage of logical operators `&&` (AND), `||`(OR) and `!(NOT)`. Examples include

`10 > 9;` // evaluates to boolean value true

`10 < 20;` // evaluates to boolean value false

`True;` //evaluates to boolean value true

`A===20 && b===30;` // evaluates to true or false based on the values of a and b

- **Primary Expressions:**

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values. Examples include the following

`'hello world';` // A string literal

`23;` // A numeric literal

`True;` // Boolean value true

`Sum;` // Value of variable sum

`This;` // A keyword that evaluates to the current object

- **Left-hand-side Expressions:**

Also known as lvalues, left-hand-side expressions are those that can appear on the left side of an assignment expression. Examples of left-hand-side expressions include the following

// variables such as I and total

`I = 10;`

`Total = 0;`

```
// properties of objects
```

```
Varobj = {}; // an empty object with no properties
```

```
Obj.x = 10; // an assignment expression
```

```
// elements of arrays
```

```
Array[0] = 20;
```

```
Array[1] = 'hello';
```

```
// Invalid left-hand-side errors
```

```
++(a+1); // SyntaxError. Attempting to increment or decrement an expression that is not an lvalue will lead to errors.
```

Now that we have covered the basics of expressions, let's dive a bit deeper into expressions.

- **Assignment Expressions:**

When expressions use the = operator to assign a value to a variable, it is called an assignment expression. Examples include

```
Average = 55;
```

```
Var b = (a = 1); // here the assignment expression (a = 1) evaluates to a value that is assigned to the variable b. b = (a = 1) is another assignment expression. Var is not part of the expression.
```

The = operator expects an lvalue as its left-side operand. The value of an assignment expression is the value of the right-side operand such as 55 in the above example. As a side effect, the = operator assigns the value on the right side to the value on the left side.

- **Expressions with side effects:**

As we just saw with assignment expressions, expressions with side effects are those that result in a change or a side effect such as setting or modifying the value of a variable through the assignment operator =, function call, incrementing or decrementing the value of a variable.

```
Sum = 20; // here sum is assigned the value of 20
```

```
Sum++; // increments the value of sum by 1
```

```
Function modify(){
```

```
    A *= 10;
```

```
}
```

```
Var a = 10;
```

```
Modify(); // modifies the value of a to 100.
```

2.7 JavaScript Loops

The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array. There are four types of loops in JavaScript.

1. For loop
2. While loop
3. Do-while loop
4. For-in loop

2.7.1 JavaScript For loop

The JavaScript for loop iterates the elements for the fixed number of times. It should be used if number of iteration is known. The syntax of for loop is given below.

```
ForFor (initialization; condition; increment)
{
Code to be executed
}
```

Let's see the simple example of for loop in javascript.

```
<script>
For (i=1; i<=5; i++)
{
Document.write(I + "<br/>")
}
</script>
```

Output:

```
1
2
3
4
5
```


2.7.2. JavaScript while loop

The JavaScript while loop iterates the elements for the infinite number of times. It should be used if number of iteration is not known. The syntax of while loop is given below.

While (condition)

```
{
```

```
    Code to be executed
```

```
}
```

Let's see the simple example of while loop in javascript.

```
<script>
```

```
Vari=11;
```

```
While (i<=15)
```

```
{
```

```
Document.write(I + "<br/>");
```

```
i++;
```

```
}
```

```
</script>
```

Test it Now

Output:

11

12

13

14

15

2.7.3 JavaScript do while loop

The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false. The syntax of do while loop is given below.

```
Do{  
    Code to be executed  
}while (condition);
```

Let's see the simple example of do while loop in javascript.

```
<script>  
Vari=21;  
Do{  
Document.write(I + "<br/>");  
I++;  
}while (i<=25);  
</script>
```

Output:

```
21  
22  
23  
24  
25
```

2.7.4 JavaScript for in loop

The JavaScript for in loop is used to iterate the properties of an object. We will discuss about it later. The for loop is the most compact form of looping and includes the following three important parts –The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out. The iteration statement where you can increase or decrease your counter.

Syntax

```
For (initialization; test condition; iteration statement){
```

Statement(s) to be executed if test condition is true

}

2.8 JavaScript Control structures

The control structures within JavaScript allow the program flow to change within a unit of code or function. These statements can determine whether or not given statements are executed – and provide the basis for the repeated execution of a block of code.

- Conditionals (if-else, switch) that perform different actions depending on the value of an expression,
- Loops (while, do-while, for, for-in, for-of), that execute other statements repetitively,
- Jumps (break, continue, labeled statement) that cause a jump to another part of the program.
 - Conditional Branches, which we use for choosing between two or more paths.
 - Loops that are used to iterate through multiple values/objects and repeatedly run specific code blocks.

If ... else

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

```
if(expression){
Statement(s) to be executed if expression istrue
}
```

Example

```
<scripttype="text/javascript">
<!--
var age =20;
if( age >18){
```

```
document.write("<b>Qualifies for driving</b>");  
}  
//-->  
</script>
```

Example #2

```
if (5 > 6) {  
document.write ("Condition is true : 5 is less than 6");  
}  
else {  
document.write ("Condition is false : 5 is not greater than 6");  
}
```

Switch case

The basic syntax of the switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax

```
switch(expression){  
case condition 1: statement(s)  
break;  
case condition 2: statement(s)  
break;  
...  
case condition n: statement(s)  
break;  
default: statement(s)  
}
```

Example

```
<scripttype="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br/>");
switch(grade){
case'A':document.write("Good job<br/>");
break;
case'B':document.write("Pretty good<br/>");
break;
case'C':document.write("Passed<br/>");
break;
case'D':document.write("Not so good<br/>");
break;
case'F':document.write("Failed<br/>");
break;
default:document.write("Unknown grade<br/>")
}
document.write("Exiting switch block");
//-->
</script>
```

2.9 JavaScript Constructor Function

Constructor functions technically are regular functions. There are two conventions though:

They are named with capital letter first.

They should be executed only with "new" operator.

For instance:

```
functionUser(name){
this.name= name;
```

```

this.isAdmin=false;
}
let user =newUser("Jack");

```

```

    alert(user.name);// Jack
    alert(user.isAdmin);// false

```

The function statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.

Note – Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

Syntax

Following is the syntax to create a function using Function() constructor along with the new operator.

```

<script type = “text/javascript”>
<!--
Varvariablename = new Function(Arg1, Arg2..., “Function Body”);
</script>

```

The Function() constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons. Notice that the Function() constructor is not passed any argument that specifies a name for the function it creates. The unnamed functions created with the Function() constructor are called anonymous functions.

```

<html>
<head>
<script type = “text/javascript”>
  <!--
  Varfunc = new Function(“x”, “y”, “return x*y;”);
  Function secondFunction() {
  Var result;
  Result = func(10,20);
  Document.write ( result );

```

```

    }
    //→
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type = "button" onclick = "secondFunction()" value = "Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>

```

Output

Click the following button to call the function

Call function

200

2.10 User defined function dialog box

There are three types of dialog boxes supported in JavaScript that are alert, confirm, and prompt. These dialog boxes can be used to perform specific tasks such as raise an alert, to get confirmation of an event or an input, and to get input from the user.

a) Alert Dialog box

It is used to provide a warning message to users. It is one of the most widely used dialog box in JavaScript. It has only one 'OK' button to continue and select the next task.

We can understand it by an example like suppose a textfield is mandatory to be filled out, but the user has not given any input value to that text field, then we can display a warning message by using the alert box.

Syntax

```
Alert(message);
```

Example

Let us see the demonstration of an alert dialog box by using the following example.

```
<html>
<head>
<script type="text/javascript">

    Function show() {
        Alert("It is an Alert dialog box");
    }
</script>

</head>
<body>
<center>

<h1>Hello World ☺☺</h1>

<h2>Welcome to javaTpoint</h2>

<p>Click the following button </p>

<input type="button" value="Click Me" onclick="show();" />

</center>

</body>

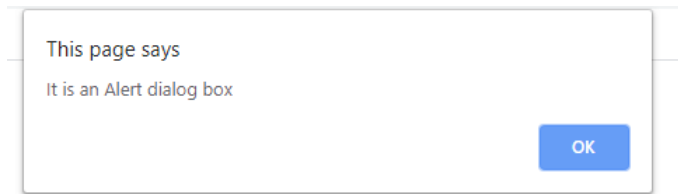
</html>
```

Output

After the successful execution of the above code, you will get the following output.

ES6 Dialog boxes

After clicking on the Click Me button, you will get the following output:



Click the following button

Click Me

b) Confirmation Dialog box

It is widely used for taking the opinion from the user on the specific option. It includes two buttons, which are **OK** and **Cancel**. As an example, suppose a user is required to delete some data, then the page can confirm it by using the confirmation box that whether he/she wants to delete it or not.

If a user clicks on the **OK** button, then the method **confirm()** returns **true**. But if the user clicks on the **cancel** button, then the **confirm() method** returns false.

Syntax

```
1. confirm(message);
```

Example

Let us understand the demonstration of this dialog box by using the following example.

```
1. <html>
2.
3. <head>
4.   <script type="text/javascript">
5.     function show() {
6.       var con = confirm ("It is a Confirm dialog box");
7.       if(con == true) {
8.         document.write ("User Want to continue");
9.       }
10.      else {
```

```
11.     document.write ("User does not want to continue");
12.     }
13.     }
14. </script>
15. </head>
16.
17. <body>
18.     <center>
19.         <h1>Hello World :) :)</h1>
20.         <h2>Welcome to javaTpoint</h2>
21.         <p>Click the following button </p>
22.         <input type="button" value="Click Me" onclick="show();" />
23.     </center>
24. </body>
25.
26. </html>
```

Output

After the successful execution of the above code, you will get the following output.

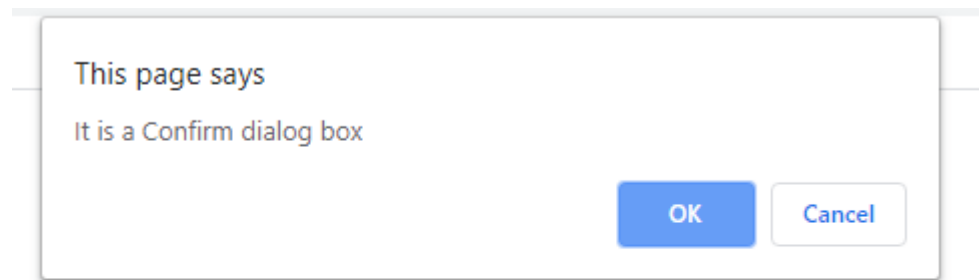
Hello World :) :)

Welcome to javaTpoint

Click the following button

Click Me

When you click on the given button, then you will get the following output:



Click the following button

Click Me

After clicking the **OK** button, you will get:

User Want to continue

On clicking the **Cancel** button, you will get:

User does not want to continue

c) Prompt Dialog box

The prompt dialog box is used when it is required to pop-up a text box for getting the user input. Thus, it enables interaction with the user.

The prompt dialog box also has two buttons, which are **OK** and **Cancel**. The user needs to provide input in the textbox and then click OK. When a user clicks on the OK button, then the dialog box reads that value and returns it to the user. But on clicking the **Cancel** button, **prompt()** method returns **null**.

Syntax

```
1. prompt(message, default_string);
```

Let us understand the prompt dialog box by using the following illustration.

Example

```
1. <html>
2.
3. <head>
4.   <script type="text/javascript">
5.     function show() {
6.       var value = prompt("Enter your Name : ", "Enter your name");
7.       document.write("Your Name is : " + value);
8.     }
9.   </script>
10. </head>
11.
12. <body>
13.   <center>
14.     <h1>Hello World :)</h1>
15.     <h2>Welcome to javaTpoint</h2>
16.     <p>Click the following button </p>
17.     <input type="button" value="Click Me" onclick="show();" />
18.   </center>
19. </body>
20.
21. </html>
```

Output

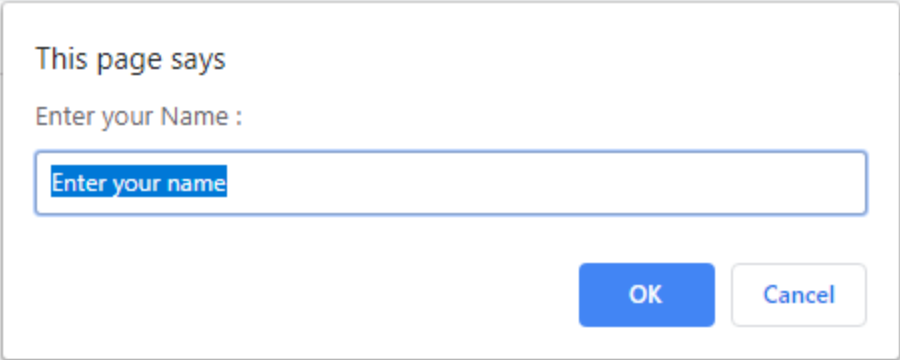
After executing the above code successfully, you will get the following output.

Hello World :) :)

Welcome to javaTpoint

Click the following button

When you click on the **Click Me** button, you will get the following output:



This page says

Enter your Name :

OK Cancel

Click Me

Enter your name and click OK button, you will get:

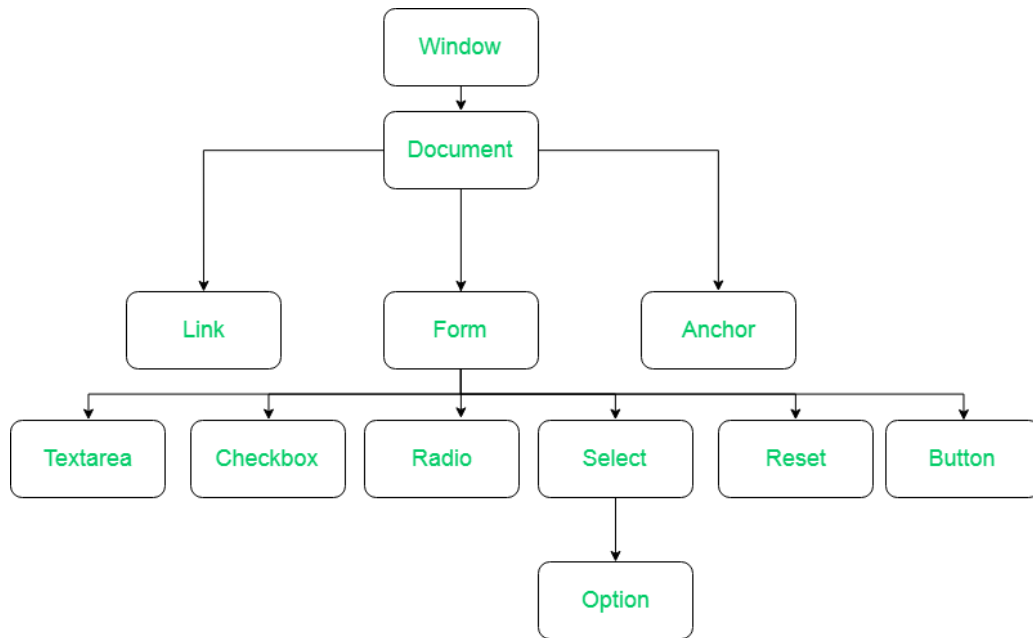
Your Name is : javaTpoint

Unit- III

3.1 JavaScript - Document Object Model or DOM

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**.



1. **Window Object:** Window Object is at always at top of hierarchy.
2. **Document object:** When HTML document is loaded into a window, it becomes a document object.
3. **Form Object:** It is represented by form tags.
4. **Link Objects:** It is represented by link tags.
5. **Anchor Objects:** It is represented by a href tags.
6. **Form Control Elements::** Form can have many control elements such as text fields, buttons, radio buttons, and checkboxes, etc.

There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

The Legacy DOM – This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.

The W3C DOM – This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

The IE4 DOM – This document object model was introduced in Version 4 of Microsoft’s Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example –

```
If (document.getElementById) {
    // If the W3C method exists, use it
} else if (document.all) {
    // If the all[] array exists, use it
} else {
    // Otherwise use the legacy DOM
}
```

3.2 HTML Object

HTML <object> tag

HTML <object> tag is used to embed multimedia files on webpage. The <object> tag can include multimedia files such as video, audio, image, PDF, Java Applets, or another page on your page.

HTML <param> tag also used with <object> tag to pass parameters to plugin which has been included with <object> tag.

If you insert text between the <object> and </object> tags, then it will only be displayed if the browser does not support the <object> tag.

Syntax

```
<object data="" type=""></object>
```

3.3 JavaScript Event Handling

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Some of the HTML events are

Clickonclick

Mouseover

Mouseout

Mousedown

Mouseup

Mousemove

Example

Try the following example.

```
<html>
<head>
<scripttype="text/javascript">
<!--
functionsayHello(){
    alert("Hello World")
}
//-->
</script>
```



```

</head>

<body>
<p>Click the following button and see result</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello"/>
</form>
</body>
</html>

```

Output

Click the following button and see result

➤ **onsubmit Event Type**

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use **onsubmit**. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```

<html>
<head>
<script type="text/javascript">
<!--
function validation(){
    all validation goes here
    .....
return either true or false

```

```

}
//-->
</script>
</head>

<body>
<formmethod="POST"action="t.cgi"onsubmit="return validate()">
    .....
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>

```

➤ **onmouseover and onmouseout**

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```

<html>
<head>
<scripttype="text/javascript">
<!--
function over(){
document.write("Mouse Over");
}
function out(){
document.write("Mouse Out");
}
//-->
</script>

```

```
</head>

<body>
<p>Bring your mouse inside the division to see the result:</p>
<divonmouseover="over()"onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

Output

Mouse Over

3.4 Window Object

The window object represents an open window in a browser. If a document contains frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

- The **window** object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object

Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

Window Size

Two properties can be used to determine the size of the browser window. Both properties return the sizes in pixels:

- `Window.innerHeight` – the inner height of the browser window (in pixels)
- `Window.innerWidth` – the inner width of the browser window (in pixels)

Methods of window object

The important methods of window object are as follows:

Method	Description
<code>alert()</code>	displays the alert box containing message with ok button.
<code>confirm()</code>	displays the confirm dialog box containing message with ok and cancel button.
<code>prompt()</code>	displays a dialog box to get input from the user.
<code>open()</code>	opens the new window.
<code>close()</code>	closes the current window.
<code>setTimeout()</code>	performs action after specified time like calling function, evaluating expressions etc.

3.5 JavaScript - Document Object

A **Document object** represents the **HTML document** that is displayed in that window. The **Document object** has various properties that refer to other **objects** which allow access to and modification of **document** content. ... **Document object** – Each **HTML document** that gets loaded into a window becomes a **document object**.

4 Browser Object

The Browser Object Model (BOM) is used to interact with the browser. The default object of browser is window means you can call all the functions of window by specifying window or directly.

The Browser Object Model (BOM) is the core of JavaScript on the web. The BOM provides you with objects that expose the web browser's functionality

For example:

```
Window.alert("hello javatpoint");
```

Is same as:

```
Alert("hello javatpoint");
```

3.6 Form Object

Form object represents an HTML form.

Form object is a Browser object of JavaScript used to access an HTML form. If a user wants to access all forms within a document then he can use the forms array. The form object is actually a property of document object that is uniquely created by the browser for each form present in a document.

- The properties and methods associated with form object are used to access the form fields, attributes and controls associated with forms.
- It is used to collect user input through elements like text fields, check box and radio button, select option, text area, submit buttons and etc.

Syntax:

```
<form> . . . </form>
```

Form Object Properties

Property	Description
Action	It sets and returns the value of the action attribute in a form.
enctype	It sets and returns the value of the enctype attribute in a form.
Length	It returns the number of elements in a form.
Method	It sets and returns the value of the method attribute in a form that is GET or POST.
Name	It sets and returns the value of the name attribute in a form.
Target	It sets and returns the value of the target attribute in a form.

Form Object Methods

Method	Description
reset()	It resets a form.
submit()	It submits a form.

3.7 JavaScript Navigator Object

The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc. The navigator object contains information about the browser.

Note: There is no public standard that applies to the navigator object, but all major browsers support it.

The navigator object is the window property, so it can be accessed by:

Window.navigator

Or,

Navigator

Property	Description
<u>appName</u>	Returns the code name of the browser
<u>appVersion</u>	Returns the name of the browser
<u>cookieEnabled</u>	Returns the version information of the browser
<u>geolocation</u>	Determines whether cookies are enabled in the browser
<u>language</u>	Returns a Geolocation object that can be used to locate the user's position
<u>onLine</u>	Returns the language of the browser
<u>platform</u>	Determines whether the browser is online
<u>product</u>	Returns for which platform the browser is compiled
<u>userAgent</u>	Returns the engine name of the browser
	Returns the user-agent header sent by the browser to the server

3.8 Screen Object

The screen object contains information about the visitor's screen. JavaScript Screen is a built-in Interface (object type) that is used to fetch information related to the browser screen on which the current window is rendered.

The window.screen object can be written without the window prefix.

Properties:

Screen.width

Screen.height

Screen.availWidth

Screen.availHeight

Screen.colorDepth

Screen.pixelDepth

Note: There is no public standard that applies to the screen object, but all major browsers support it.

Screen Object Properties

Property	Description
<u>availHeight</u>	Returns the height of the screen (excluding the Windows Taskbar)
<u>availWidth</u>	Returns the width of the screen (excluding the Windows Taskbar)
<u>colorDepth</u>	Returns the bit depth of the color palette for displaying images
<u>height</u>	Returns the total height of the screen
<u>pixelDepth</u>	Returns the color resolution (in bits per pixel) of the screen
<u>width</u>	Returns the total width of the screen

3.9 JavaScript Built-in Objects

Built-in objects are not related to any Window or DOM object model. These objects are used for simple data processing in the JavaScript. Built-in objects: which are provided by the JavaScript core. Things like Array, Strings, Number, Boolean, RegExp are all built-in objects

Math Object

Math object is a built-in static object.

It is used for performing complex math operations.

Object	Methods	Description
Array Object	concat()	Concatenate the two or more strings.
	join()	Join the two or more array strings.
	reverse()	Reverse the array string.
	toString()	Convert the array into string array.
Object	Methods	Description
String Object	anchor()	Create an Anchor.
	big()	Big Font display into string.
	bold()	bold style display into string.
	fixed()	fixed font display into string.
	FontColor()	Font Color display into string.
	FontSize()	Font Size display into string.

italic()	italic style display into string.
link()	link display into string.
small()	small text display into string.
strike()	strike text display into string.
sub()	sub style display into string.
sup()	sup style display into string.
toLowerCase	toLowerCase text display into string.
toUpperCase	toUpperCase text display into string.

3.10 User-defined objects:

User-defined object these are objects which you have created in your program or application. **User-defined object** types are the types that most developers will use to expose their services.

These types correspond most closely to classes in many **object**-oriented languages. ... Methods are also specialized to produce constructors, which can be either instance or class constructors (.ctor and .cctor).

Create() method to create an object of any prototype object which is already defined. ... As you can see in the code above, we can use the Object. Create method to create a new object using an already defined object, and then if required we can change the properties values and use the object function too.

3.11 JavaScript Cookies

Cookies are data, stored in small text files, on your computer. When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem “how to remember information about the user”:

When a user visits a web page, his/her name can be stored in a cookie.

Next time the user visits the page, the cookie “remembers” his/her name.

Cookies are saved in name-value pairs like:

Username = John D

How Cookies Works?

When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.

So, to recognize the old user, we need to add the cookie with the response from the server.

Browser at the client-side.

Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.

Create a Cookie with JavaScript

A **cookie** is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.

A **cookie** contains the information as a string generally in the form of a name-value pair separated by semi-colons.

JavaScript can create, read, and delete cookies with the `document.cookie` property. With JavaScript, a cookie can be created like this:

```
Document.cookie = "username=John Doe";
```

UNIT- IV

4.1 ASP.NET LANGUAGE STRUCTURE

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

ASP.NET Web Forms Model

ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

Now, HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

- Page state
- Session state

The page state is the state of the client, i.e., the content of various input fields in the web form. The session state is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart.

User adds items to a shopping cart. Items are selected from a page, say the items page, and the total collected items and price are shown on a different page, say the cart page. Only HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.

The ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.

This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

The ASP.NET Component Model

The ASP.NET component model provides various building blocks of ASP.NET pages. Basically it is an object model, which describes:

- Server side counterparts of almost all HTML elements or tags, such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.

ASP.NET is a technology, which works on the .Net framework that contains all web-related functionalities. The .Net framework is made of an object-oriented hierarchy. An ASP.NET web application is made of pages. When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

4.2 PAGE STRUCTURE

An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images. Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.

ASP.NET runtime controls the association between a page instance and its state. An ASP.NET page is an object of the Page or inherited from it.

All the controls on the pages are also objects of the related control class inherited from a parent Control class. When a page is run, an instance of the object page is created along with all its content controls.

An ASP.NET page is also a server side file saved with the .aspx extension. It is modular in nature and can be divided into the following core sections:

- Page Directives
- Code Section
- Page Layout

Page Directives

The page directives set up the environment for the page to run. The @Page directive defines page-specific attributes used by ASP.NET page parser and compiler. Page directives specify how the page should be processed, and which assumptions need to be taken about the page.

It allows importing namespaces, loading assemblies, and registering new controls with custom tag names and namespace prefixes.

Code Section

The code section provides the handlers for the page and control events along with other functions required. We mentioned that, ASP.NET follows an object model. Now, these objects raise events when some events take place on the user interface, like a user clicks a button or moves the cursor. The kind of response these events need to reciprocate is coded in the event handler functions. The event handlers are nothing but functions bound to the controls.

The code section or the code behind file provides all these event handler routines, and other functions used by the developer. The page code could be precompiled and deployed in the form of a binary assembly.

Page Layout

The page layout provides the interface of the page. It contains the server controls, text, inline JavaScript, and HTML tags.

The following code snippet provides a sample ASP.NET page explaining Page directives, code section and page layout written in C#:

```
<!-- directives -->  
<% @PageLanguage="C#" %>
```

```
<!-- code section -->
<script runat="server">

private void convertToUpper(object sender, EventArgs e)
{
    string str=mytext.Value;
    changed_text.InnerHtml=str.ToUpper();
}
</script>

<!-- Layout -->
<html>
<head>
<title> Change to Upper Case </title>
</head>

<body>
<h3> Conversion to Upper Case </h3>

<form runat="server">
<input runat="server" id="mytext" type="text"/>
<input runat="server" id="button1" type="submit" value="Enter..." OnServerClick="convertToUpper"
"/>

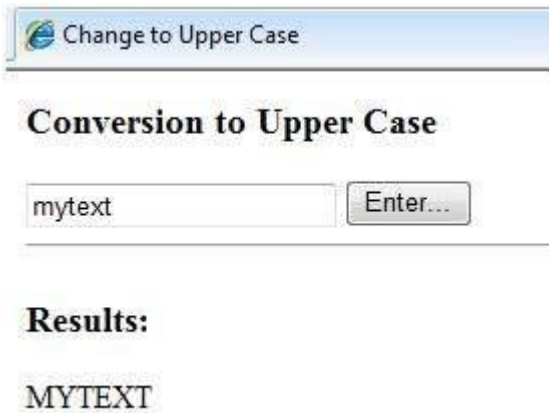
<hr/>
<h3> Results: </h3>
<span runat="server" id="changed_text"/>
</form>

</body>
```



```
</html>
```

Copy this file to the web server root directory. Generally it is c:\inetput\wwwroot. Open the file from the browser to execute it and it generates following result:



Using Visual Studio IDE

Let us develop the same example using Visual Studio IDE. Instead of typing the code, you can just drag the controls into the design view:



The content file is automatically developed. All you need to add is the Button1_Click routine, which is as follows:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string buf = TextBox1.Text;
    changed_text.InnerHtml = buf.ToUpper();
}
```

```
}

```

The content file code is as given:

```
<% @PageLanguage="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="firstexample._Default" %>

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

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

<head runat="server">
<title>
    Untitled Page
</title>
</head>

<body>

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

<asp:TextBox ID="TextBox1" runat="server" style="width:224px">
</asp:TextBox>

<br/>
<br/>

<asp:Button ID="Button1" runat="server" Text="Enter..." style="width:85px" onclick="Button1_Click"/>

```

```
<hr/>

<h3> Results: </h3>
<spanrunat="server" id="changed_text"/>

</div>
</form>

</body>

</html>
```

Execute the example by right clicking on the design view and choosing 'View in Browser' from the popup menu. This generates the following result:



4.3 PAGE EVENT

ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

- **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
- **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.
- **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
- **LoadViewState** - LoadViewState event allows loading view state information into the controls.
- **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
- **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
- **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
- **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler
- **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
- **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
- **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.

- **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

4.4 ASP.NET DIRECTIVES

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

The syntax for declaring a directive is:

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

In this section, we will just introduce the ASP.NET directives and we will use most of these directives throughout the tutorials.

The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file.

The basic syntax of Application directive is:

```
<% @ApplicationLanguage="C#" %>
```

The attributes of the Application directive are:

Attributes	Description
Inherits	The name of the class from which to inherit.
Description	The text description of the application.

	Parsers and compilers ignore this.
Language	The language used in code blocks.

The Assembly Directive

The Assembly directive links an assembly to the page or the application at parse time. This could appear either in the global.asax file for application-wide linking, in the page file, a user control file for linking to a page or user control.

The basic syntax of Assembly directive is:

```
<% @AssemblyName="myassembly" %>
```

The attributes of the Assembly directive are:

Attributes	Description
Name	The name of the assembly to be linked.
Src	The path to the source file to be linked and compiled dynamically.

The Control Directive

The control directive is used with the user controls and appears in the user control (.ascx) files.

The basic syntax of Control directive is:

```
<% @ControlLanguage="C#"EnableViewState="false" %>
```

The attributes of the Control directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables automatic association of events to handlers.
ClassName	The file name for the control.
Debug	The Boolean value that enables or disables compiling with debug symbols.
Description	The text description of the control page, ignored by compiler.
EnableViewState	The Boolean value that indicates whether view state is maintained across page requests.
Explicit	For VB language, tells the compiler to use option explicit mode.
Inherits	The class from which the control page inherits.

Language	The language for code and script.
Src	The filename for the code-behind class.
Strict	For VB language, tells the compiler to use the option strict mode.

The Implements Directive

The Implement directive indicates that the web page, master page or user control page must implement the specified .Net framework interface.

The basic syntax for implements directive is:

```
<% @ImplementsInterface="interface_name" %>
```

The Import Directive

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

The basic syntax for import directive is:

```
<% @namespace="System.Drawing" %>
```

The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

```
<% @MasterPageLanguage="C#" AutoEventWireup="true" CodeFile="SiteMater.master.cs" Inherits="SiteMaster" %>
```


The MasterType Directive

The MasterType directive assigns a class name to the Master property of a page, to make it strongly typed.

The basic syntax of MasterType directive is:

```
<% @MasterType attribute="value"[attribute="value"...] %>
```

The OutputCache Directive

The OutputCache directive controls the output caching policies of a web page or a user control.

The basic syntax of OutputCache directive is:

```
<% @OutputCacheDuration="15"VaryByParam="None" %>
```

The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

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

The attributes of the Page directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page_Load.
Buffer	The Boolean value that enables or disables HTTP response

	buffering.
ClassName	The class name for the page.
ClientTarget	The browser for which the server controls should render content.
CodeFile	The name of the code behind file.
Debug	The Boolean value that enables or disables compilation with debug symbols.
Description	The text description of the page, ignored by the parser.
EnableSessionState	It enables, disables, or makes session state read-only.
EnableViewState	The Boolean value that enables or disables view state across page requests.
ErrorPage	URL for redirection if an unhandled page exception occurs.
Inherits	The name of the code behind or other class.
Language	The programming language for code.
Src	The file name of the code behind class.
Trace	It enables or disables tracing.
TraceMode	It indicates how trace messages are displayed, and sorted by time or category.
Transaction	It indicates if transactions are supported.

ValidateRequest	The Boolean value that indicates whether all input data is validated against a hardcoded list of values.
-----------------	--

The PreviousPageType Directive

The PreviousPageType directive assigns a class to a page, so that the page is strongly typed.

The basic syntax for a sample PreviousPagetype directive is:

```
<% @PreviousPageType attribute="value"[attribute="value"...] %>
```

The Reference Directive

The Reference directive indicates that another page or user control should be compiled and linked to the current page.

The basic syntax of Reference directive is:

```
<% @ReferencePage="somepage.aspx" %>
```

The Register Directive

The Register derivative is used for registering the custom server controls and user controls.

The basic syntax of Register directive is:

```
<% @RegisterSrc="~/footer.ascx"TagName="footer"TagPrefix="Tfooter" %>
```

4.5 HTML SERVER CONTROLS

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing. The HTML controls such as the header tags, anchor tags, and input elements are not processed by the server but are sent to the browser for display.

They are specifically converted to a server control by adding the attribute `runat="server"` and adding an id attribute to make them available for server-side processing.

For example, consider the HTML input control:

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

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

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

Advantages of using HTML Server Controls

Although ASP.NET server controls can perform every job accomplished by the HTML server controls, the later controls are useful in the following cases:

- Using static tables for layout purposes.
- Converting a HTML page to run under ASP.NET

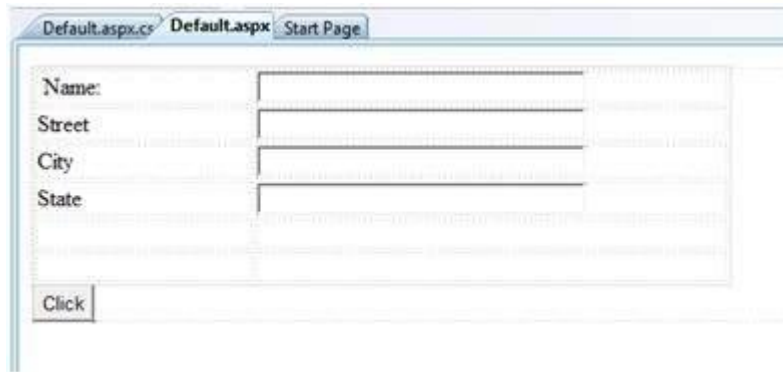
The following table describes the HTML server controls:

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

Example

The following example uses a basic HTML table for layout. It uses some boxes for getting input from the users such as name, address, city, state etc. It also has a button control, which is clicked to get the user data displayed in the last row of the table.

The page should look like this in the design view:



The code for the content page shows the use of the HTML table element for layout.

```
<% @PageLanguage="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="ht
mlserver._Default" %>

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

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

<head runat="server">

<title>Untitled Page</title>

<style type="text/css">

.style1

{
```

```
width:156px;
}
.style2
{
width:332px;
}
</style>
</head>
<body>
<formid="form1"runat="server">
<div>
<tablestyle="width:54%;">
<tr>
<tdclass="style1">Name:</td>
<tdclass="style2">
<asp:TextBoxID="txtname"runat="server"style="width:230px">
</asp:TextBox>
```

```
</td>

</tr>

<tr>

<tdclass="style1">Street</td>

<tdclass="style2">

<asp:TextBoxID="txtstreet"runat="server"style="width:230px">

</asp:TextBox>

</td>

</tr>

<tr>

<tdclass="style1">City</td>


<tdclass="style2">

<asp:TextBoxID="txtcity"runat="server"style="width:230px">

</asp:TextBox>

</td>

</tr>
```



```
<tr>

<tdclass="style1">State</td>

<tdclass="style2">

<asp:TextBoxID="txtstate"runat="server"style="width:230px">

</asp:TextBox>

</td>

</tr>

<tr>

<tdclass="style1"> </td>

<tdclass="style2"></td>

</tr>

<tr>

<tdclass="style1"></td>

<tdID="displayrow"runat="server"class="style2">

</td>

</tr>

</table>

</div>
```



```

<asp:ButtonID="Button1"runat="server"onclick="Button1_Click"Text="Click"/>

</form>

</body>

</html>

```

The code behind the button control:

```

protectedvoidButton1_Click(object sender,EventArgs e)
{
stringstr="";
str+=txtname.Text+"<br />";
str+=txtstreet.Text+"<br />";
str+=txtcity.Text+"<br />";
str+=txtstate.Text+"<br />";

displayrow.InnerHtml=str;
}

```

ANCHOR

The **Anchor** Tag Helper adds the hash character (#). It is useful with client-side application. It can be used to easy marking and searching in JavaScript. It is used to set the area name whichASP.NET Core uses to set an appropriate route.

TABLE

A table Web control creates an HTML table in simple HTML with the help of the <tr> and <td> tags.

You can use the <table>, <tr>, and <td> tags to create a table and its rows in HTML. For example, the HTML code in Listing 7-25 creates a table with two rows and three columns with their values.

Listing. HTML code for a Table control

```
1. <table border="1" width="39%">
2.   <tr>
3.     <td width="33%">
4.       Row1,Col1
5.     </td>
6.     <td width="33%">
7.       Row1,Col2
8.     </td>
9.     <td width="34%">
10.      Row1,Col3
11.    </td>
12.  </tr>
13.  <tr>
14.    <td width="33%">
15.      Row2,Col1
16.    </td>
17.    <td width="33%">
18.      Row2,Col2
19.    </td>
20.    <td width="34%">
21.      Row2,Col3
22.    </td>
23.  </tr>
24. </table>
```

In the .NET Framework the Table class enables you to build an HTML table.

The System.Web.UI.Controls namespace defines the Table class, along with the other Web controls. You can create tables in .NET using a Table control and its helper controls TableRow and TableCell. As with all Web controls, you can create a Table control at run-time as well as at design-time using the VS .NET IDE. Table 7-9 describes the Table control and its helper controls.

Forms

ASP.NET Web Forms is a part of the **ASP.NET** web application framework and is included with Visual Studio. ... Web **Forms** are pages that your users request using their browser. These pages can be written using a combination of HTML, client-script, server controls, and server code.

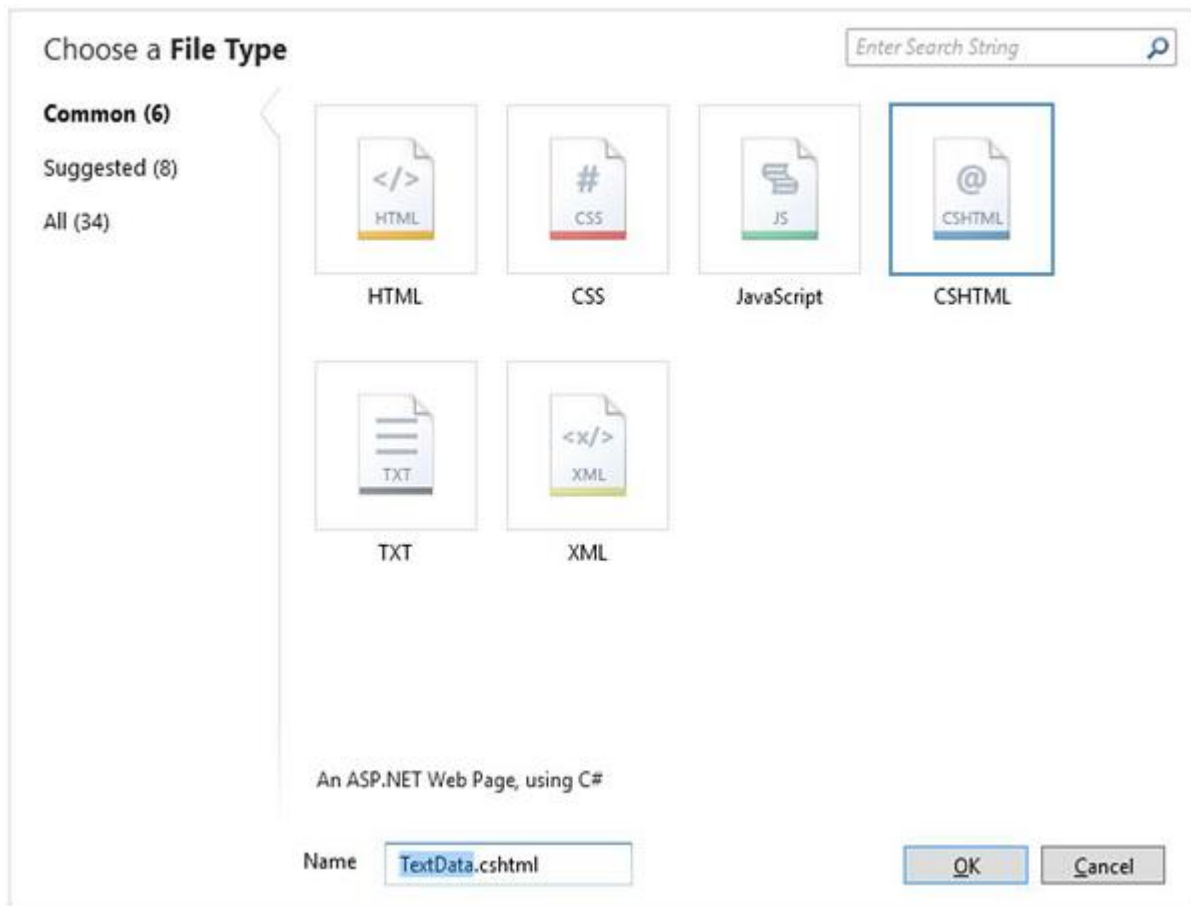
Files

files as a simple way to store data for your website.

- Text files can be in different formats, such as *.txt, *.xml, or *.csv.
- You can use the **File.WriteAllText** method to specify the file to create and then write data to it.
- You can read/write and move data from/to the text file.

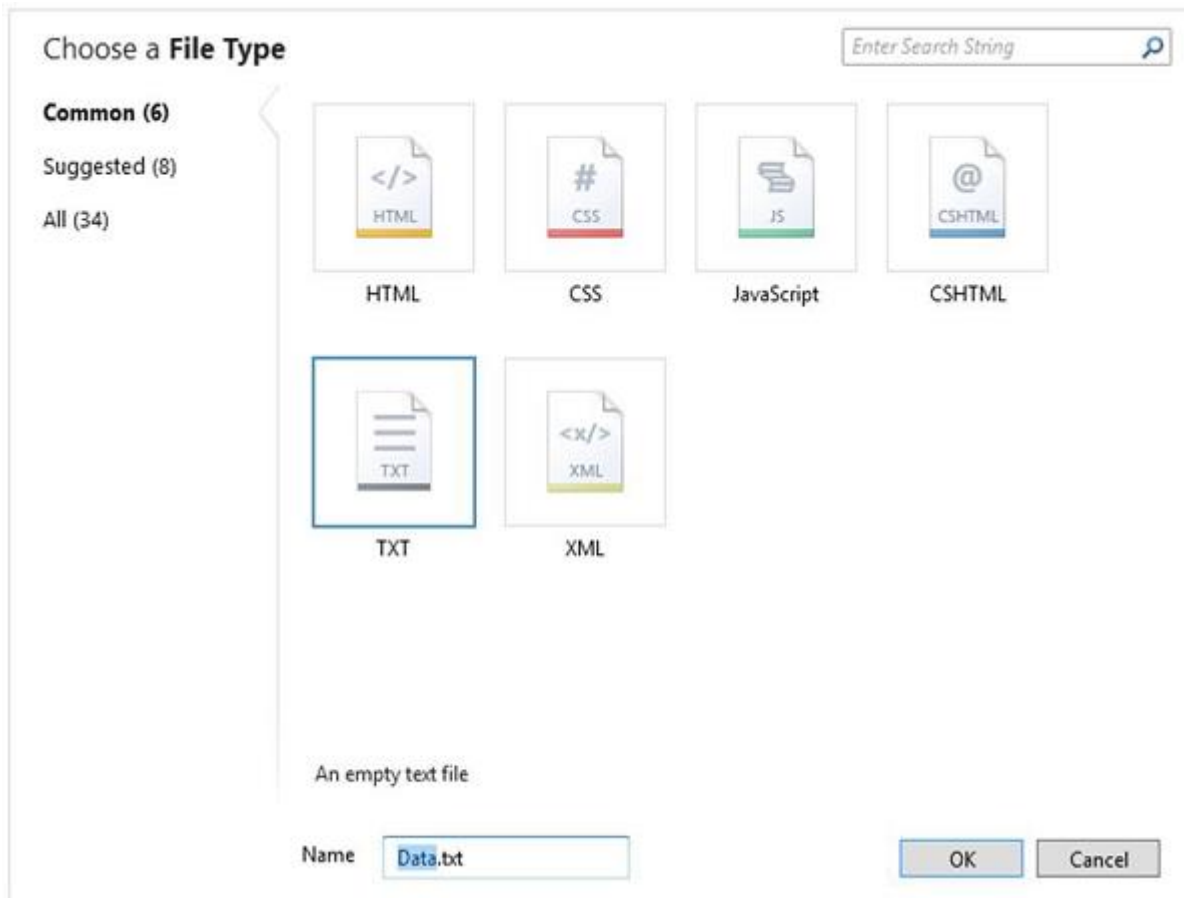
Write Data to a File

Let's have a look into a simple example in which we will write a student information into a text file. First we need to create a new CSHTML file



Enter **TextData.cshtml** in the name field and click OK to continue. In this example, we will create a simple form in which the user can enter Student information like first name, last name and marks.

We also need to create a text file in the **App_Data** folder with **Data.txt** name



Let's replace the following code in the **TextData.cshtml** file.

```
@{  
  
var result ="";  
  
if(IsPost){  
  
var firstName=Request["FirstName"];  
  
var lastName=Request["LastName"];  
  
var marks =Request["Marks"];  
  
var userData=firstName+","+lastName+","+ marks +Environment.NewLine;  
  
vardataFile=Server.MapPath("~/App_Data/Data.txt");
```

```
File.WriteAllText(@dataFile,userData);

    result ="Information saved.";
}
}

<!DOCTYPE html>

<html>

<head>

<title>WriteData to a File</title>

</head>

<body>

<form id ="form1" method ="post">

<div>

<table>

<tr>

<td>FirstName:</td>

<td><input id ="FirstName" name ="FirstName" type ="text"/></td>

</tr>

<tr>

<td>LastName:</td>
```

```
<td><input id="LastName" name="LastName" type="text"/></td>
```

```
</tr>
```

```
<tr>
```

```
<td>Marks:</td>
```

```
<td><input id="Marks" name="Marks" type="text"/></td>
```

```
</tr>
```

```
<tr>
```

```
<td></td>
```

```
<td><input type="submit" value="Submit"/></td>
```

```
</tr>
```

```
</table>
```

```
</div>
```

```
<div>
```

```
@if(result != ""){
```

```
<p>Result: @result</p>
```

```
}
```

```
</div>
```

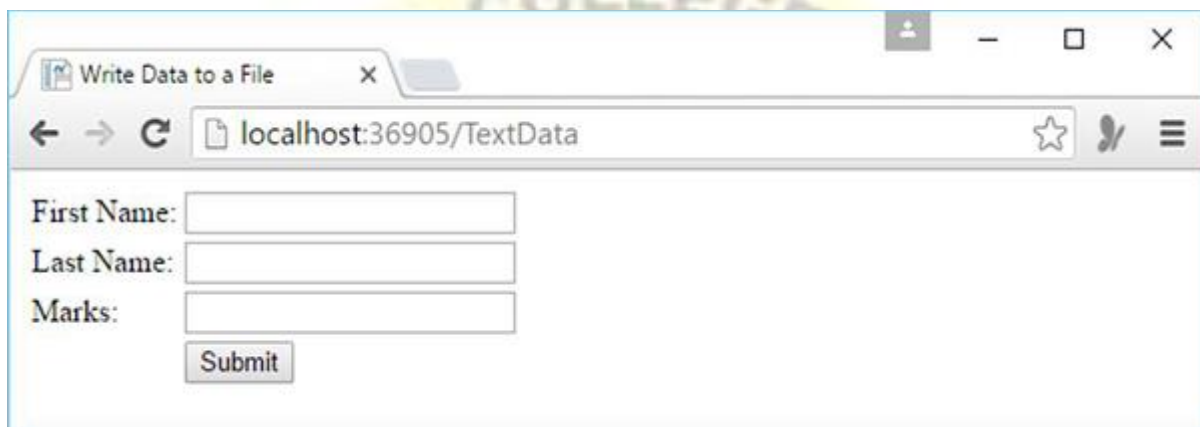
```
</form>
```

```
</body>
```

```
</html>
```

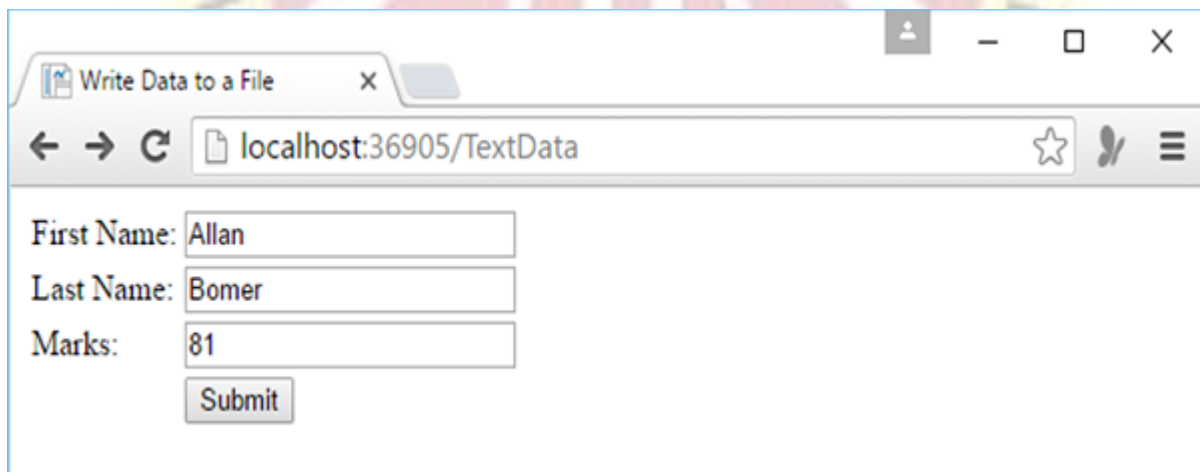
In the code, we have used the **IsPost** property to determine whether the page has been submitted before it starts processing. The **WriteAllText** method of the File object takes two parameters, the file name path and the actual data to write to the file.

Now let's run this application and specify the following url – **http://localhost:36905/TextData** and you will see the following web page.



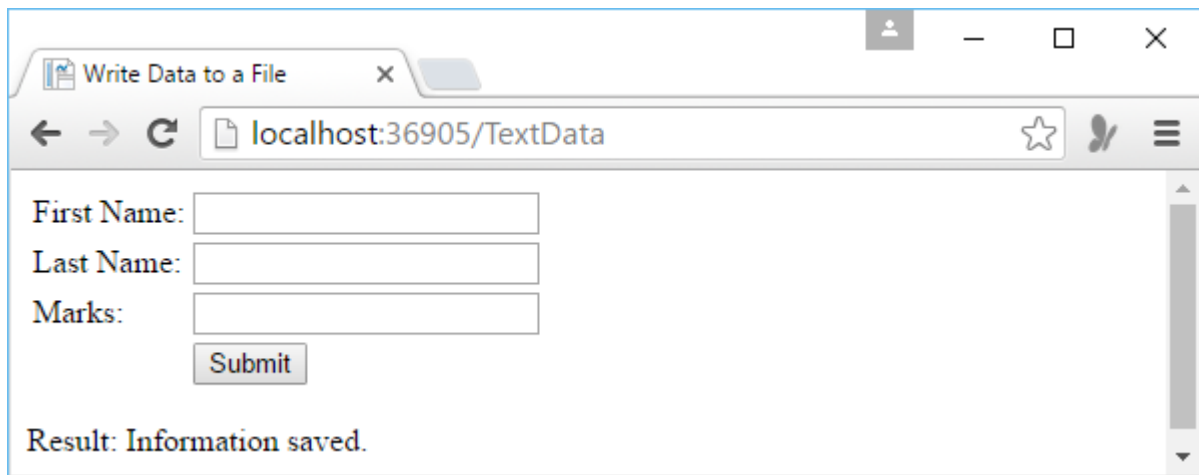
The screenshot shows a web browser window with the title "Write Data to a File" and the address bar displaying "localhost:36905/TextData". The page content includes three text input fields labeled "First Name:", "Last Name:", and "Marks:". Below these fields is a "Submit" button.

Let's enter some data in all the fields.



The screenshot shows the same web browser window as before, but now the input fields contain data: "First Name: Allan", "Last Name: Bomer", and "Marks: 81". The "Submit" button remains below the fields.

Now click on the submit button.



Write Data to a File

localhost:36905/TextData

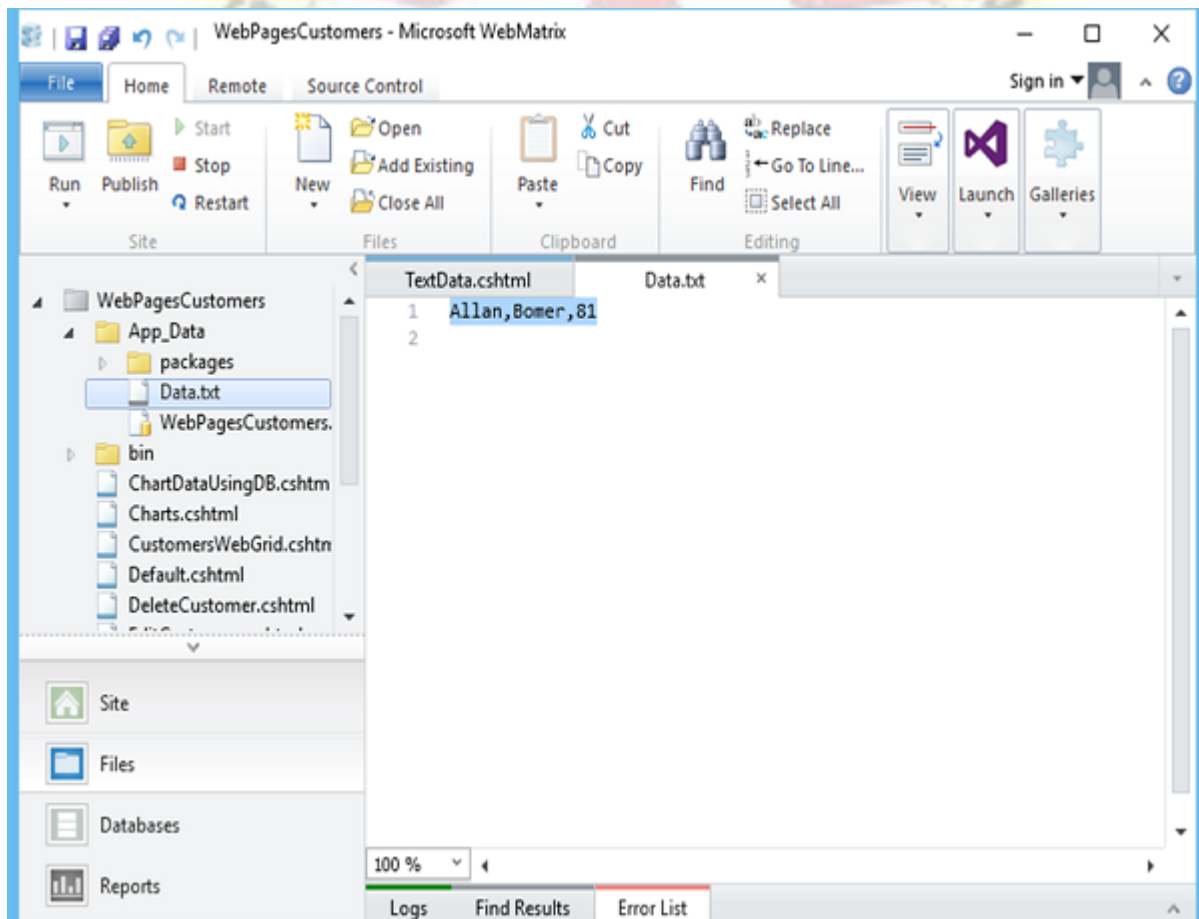
First Name:

Last Name:

Marks:

Result: Information saved.

As you can see the information is saved, now let's open the **Data.txt** file and you will see that data is written to the file.



Append Data to an Existing File

For writing data to the text file we have used `WriteAllText`. If you call this method again and pass it with the same file name, then it will overwrite the existing file completely. But in most cases, we often want to add new data to the end of the file, so we can do that by using the **`AppendAllText`** method of the file object.

Let's have a look into the same example, we will just change the **`WriteAllText()`** to **`AppendAllText ()`** as shown in the following program.

```
@{
var result = "";

if(IsPost){
varfirstName=Request["FirstName"];
varlastName=Request["LastName"];
var marks =Request["Marks"];
varuserData=firstName+","+lastName+","+ marks +Environment.NewLine;
vardataFile=Server.MapPath("~/App_Data/Data.txt");
File.AppendAllText(@dataFile,userData);
    result = "Information saved.";
}
}

<!DOCTYPE html>

<html>
```

```
<head>

<title>WriteData to a File</title>

</head>

<body>

<form id ="form1" method ="post">

<div>

<table>

<tr>

<td>FirstName:</td>

<td><input id ="FirstName" name ="FirstName" type ="text"/></td>

</tr>

<tr>

<td>LastName:</td>

<td><input id ="LastName" name ="LastName" type ="text"/></td>

</tr>

<tr>

<td>Marks:</td>

<td><input id ="Marks" name ="Marks" type ="text"/></td>
```

```

</tr>

<tr>

<td></td>

<td><input type ="submit" value="Submit"/></td>

</tr>

</table>

</div>

<div>

@if(result != ""){

<p>Result: @result</p>

}

</div>

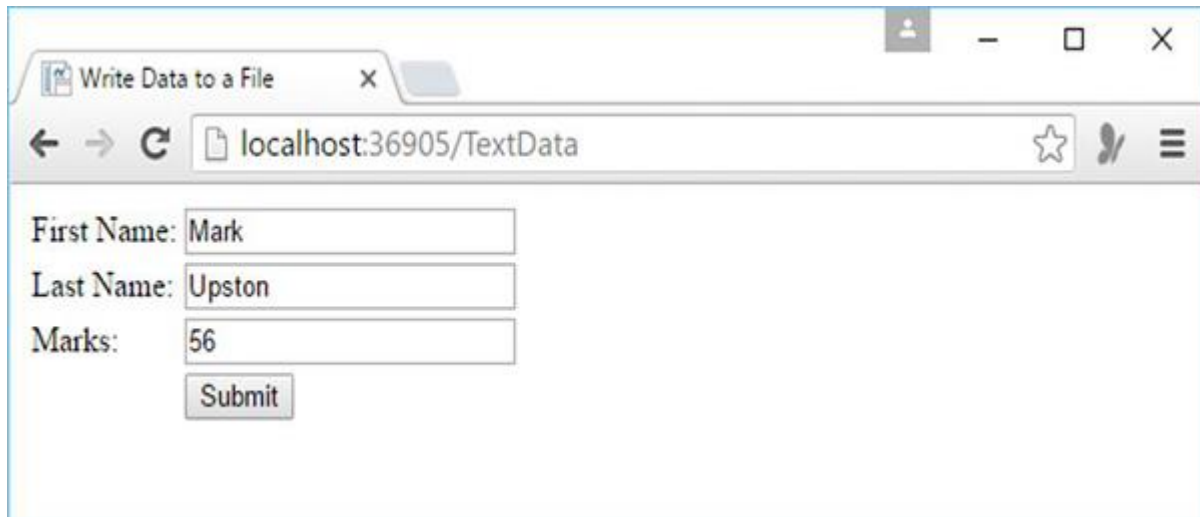
</form>

</body>

</html>

```

Now let's run the application and specify the following url <http://localhost:36905/TextData> and you will see the following web page.



Write Data to a File

localhost:36905/TextData

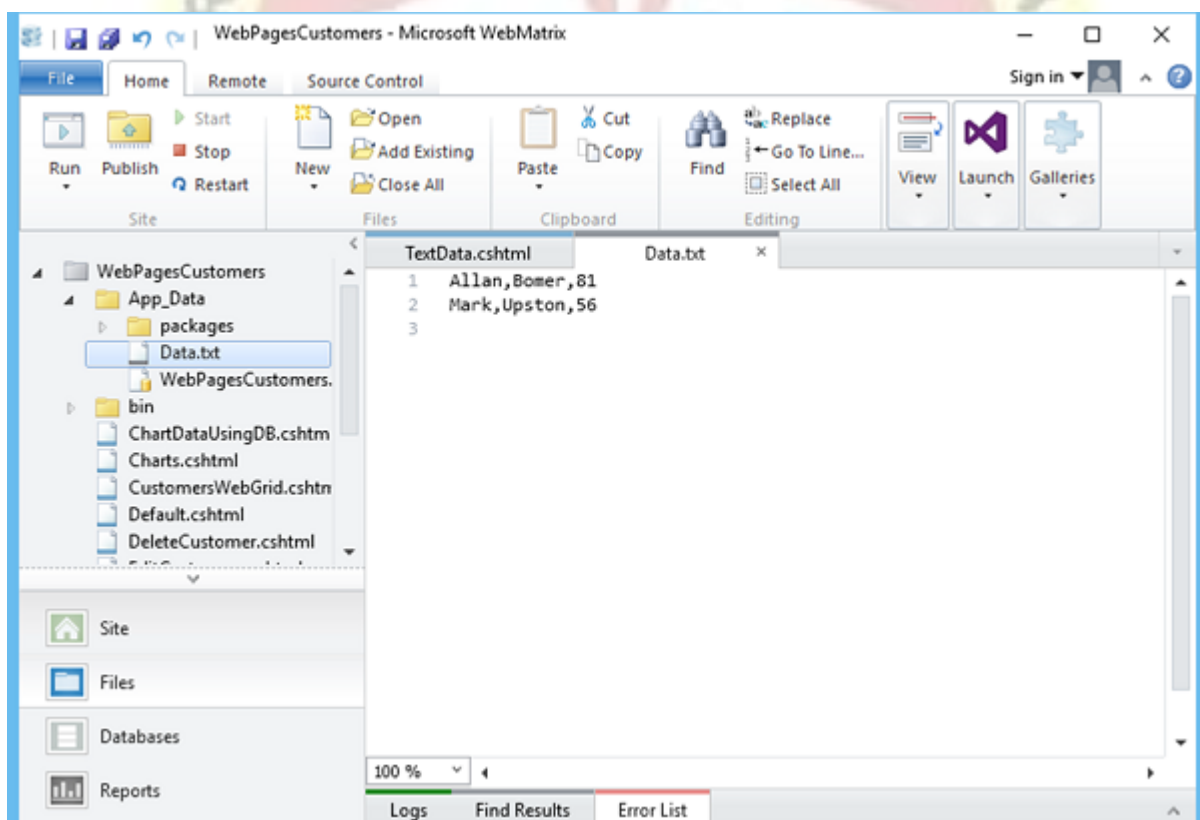
First Name:

Last Name:

Marks:

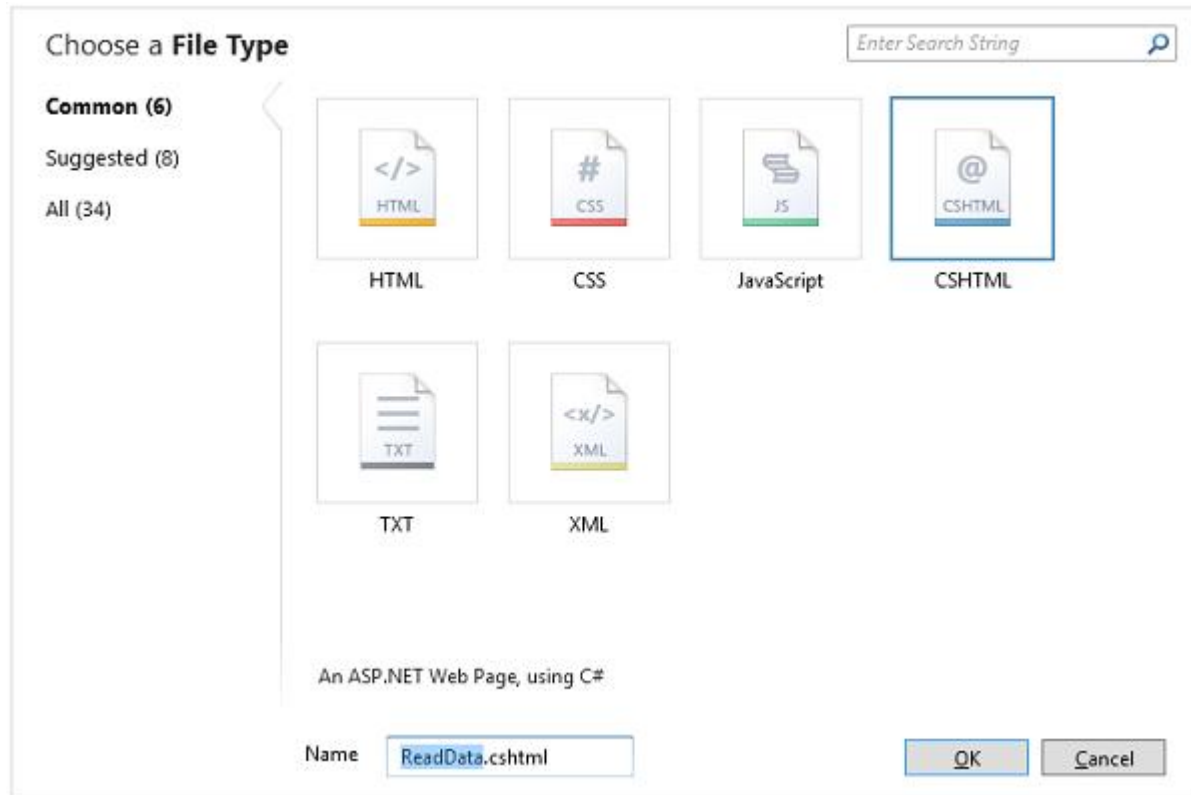
Enter some data and click the submit button.

Now when you open the Data.txt file then you will see that the data is appended at the end of this file.



Read Data from a File

To read the data from a file, you can use the File object and then call **ReadAllLines()**, which will read all the lines from the file. To do so, let's create a new CSHTML file.



Enter **ReadData.cshtml** in the Name field and click OK.

Now replace the following code in the ReadData.cshtml file.

```
@{
var result = "";
ArrayuserData=null;
char[]delimiterChar={';'};
vardataFile=Server.MapPath("~/App_Data/Data.txt");

if(File.Exists(dataFile)){
```

```
userData=File.ReadAllLines(dataFile);

if(userData==null){

// Empty file.

    result ="The file is empty.";

}

}else{

// File does not exist.

    result ="The file does not exist.";

}

}

<!DOCTYPE html>

<html>

<head>

<title>ReadingDatafrom a File</title>

</head>

<body>

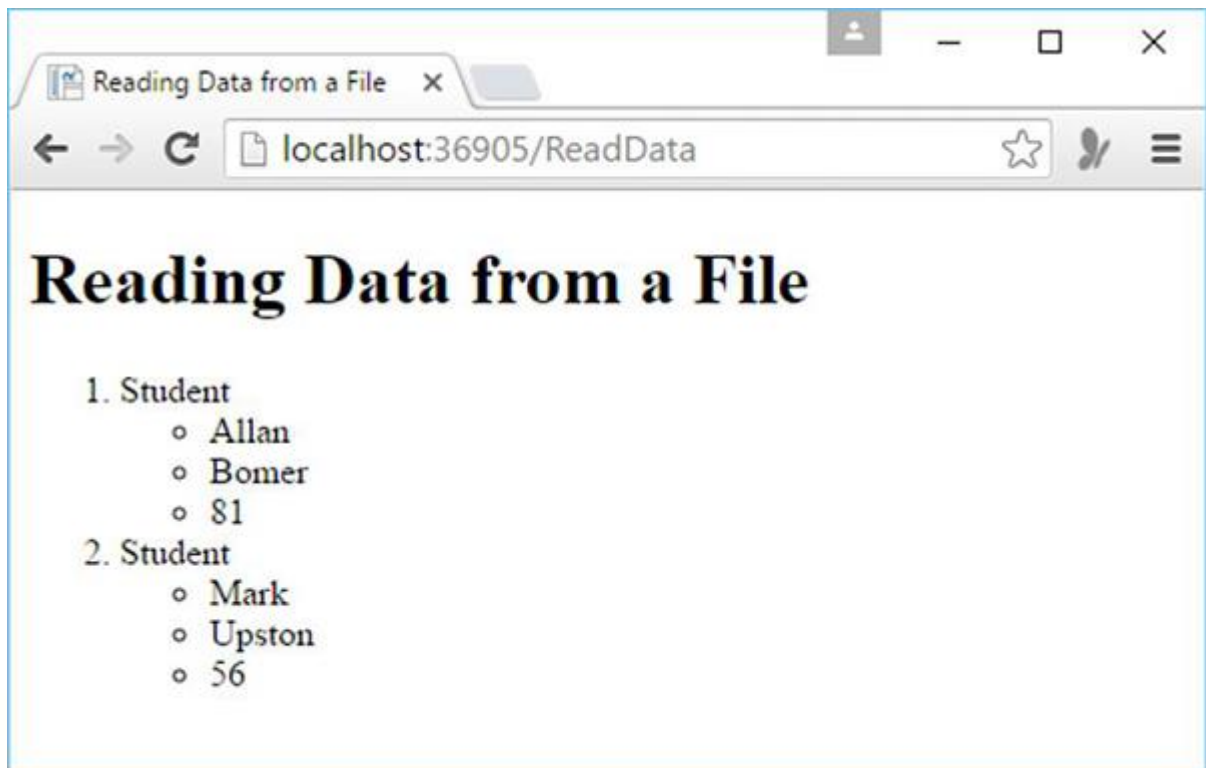
<div>

<h1>ReadingDatafrom a File</h1>

@result
```

```
@if(result == ""){  
  
<ol>  
  
@foreach(stringdataLineinuserData){  
  
<li>  
  
Student  
  
<ul>  
  
@foreach(stringdataItemindataLine.Split(delimiterChar)){  
  
<li>@dataItem</li >  
  
}  
  
</ul>  
  
</li>  
  
}  
  
</ol>  
  
}  
  
</div>  
  
</body>  
  
</html>
```

Now let's run the application again and specify the following url <http://localhost:36905/ReadData> and you will see the following web page.



4.6 BASIC WEB SERVER CONTROL

we will discuss the basic controls available in ASP.NET.

❖ Button Controls

ASP.NET provides three types of button control:

- **Button** : It displays text within a rectangular area.
- **Link Button** : It displays text that looks like a hyperlink.
- **Image Button** : It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Basic syntax of button control:

```
<asp:ButtonID="Button1"runat="server"onclick="Button1_Click"Text="Click"/>
```

Common properties of the button control:

Property	Description
Text	The text displayed on the button. This is for button and link button controls only.
ImageUrl	For image button control only. The image to be displayed for the button.
AlternateText	For image button control only. The text to be displayed if the browser cannot display the image.
CausesValidation	Determines whether page validation occurs when a user clicks the button. The default is true.
CommandName	A string value that is passed to the command event when a user clicks the button.
CommandArgument	A string value that is passed to the command event when a user clicks the button.
PostBackUrl	The URL of the page that is requested when the user clicks the button.

❖ **Text Boxes and Labels**

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

```
<asp:TextBoxID="txtstate"runat="server"></asp:TextBox>
```

Common Properties of the Text Box and Labels:

Property	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLine creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box.
MaxLength	The maximum number of characters that can be entered into the text box.
Wrap	It determines whether or not text wraps automatically for multi-line text box; default is true.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user can not change the text.
Columns	The width of the text box in characters. The actual width is determined based on the font that is used for the text entry.
Rows	The height of a multi-line text box in lines. The default value is 0, means a single line text box.

The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

❖ Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

```
<asp:CheckBoxID="chkoption"runat="Server">
</asp:CheckBox>
```

Basic syntax of radio button:

```
<asp:RadioButtonID="rdboption"runat="Server">
</asp: RadioButton>
```

Common properties of check boxes and radio buttons:

Property	Description
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

❖ List Controls

ASP.NET provides the following controls

- Drop-down list,
- List box,
- Radio button list,
- Check box list,
- Bulleted list.

These controls let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

```
<asp:ListBoxID="ListBox1"runat="server"AutoPostBack="True"OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Basic syntax of drop-down list control:

```
<asp:DropDownListID="DropDownList1"runat="server"AutoPostBack="True"OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Common properties of list box and drop-down Lists:

Property	Description
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.

Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.
SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string ("").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

Common properties of each list item objects:

Property	Description
Text	The text displayed for the item.
Selected	Indicates whether the item is selected.
Value	A string value associated with the item.

❖ **The ListItemCollection**

The ListItemCollection object is a collection of ListItem objects. Each ListItem object represents one item in the list. Items in a ListItemCollection are numbered from 0.

When the items into a list box are loaded using strings like: `lstcolor.Items.Add("Blue")`, then both the Text and Value properties of the list item are set to the string value you specify. To set it differently you must create a list item object and then add that item to the collection.

The ListItemCollection Editor is used to add item to a drop-down list or list box. This is used to create a static list of items. To display the collection editor, select edit item from the smart tag

menu, or select the control and then click the ellipsis button from the Item property in the properties window.

Common properties of ListItemCollection:

Property	Description
Item(integer)	A ListItem object that represents the item at the specified index.
Count	The number of items in the collection.

Common methods of ListItemCollection:

Methods	Description
Add(string)	Adds a new item at the end of the collection and assigns the string parameter to the Text property of the item.
Add(ListItem)	Adds a new item at the end of the collection.
Insert(integer, string)	Inserts an item at the specified index location in the collection, and assigns string parameter to the text property of the item.
Insert(integer, ListItem)	Inserts the item at the specified index location in the collection.
Remove(string)	Removes the item with the text value same as the string.
Remove(ListItem)	Removes the specified item.
RemoveAt(integer)	Removes the item at the specified index as the

	integer.
Clear	Removes all the items of the collection.
FindByValue(string)	Returns the item whose value is same as the string.
FindByValue(Text)	Returns the item whose text is same as the string.

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of radio button list:

```
<asp:RadioButtonListID="RadioButtonList1"runat="server"AutoPostBack="True"
OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

Basic syntax of check box list:

```
<asp:CheckBoxListID="CheckBoxList1"runat="server"AutoPostBack="True"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Common properties of check box and radio button lists:

Property	Description
RepeatLayout	This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

❖ **Bulleted lists and Numbered lists**

The bulleted list control creates bulleted lists or numbered lists. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of a bulleted list:

```
<asp:BulletedListID="BulletedList1"runat="server">
</asp:BulletedList>
```

Common properties of the bulleted list:

Property	Description
BulletStyle	This property specifies the style and looks of the bullets, or numbers.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

❖ **HyperLink Control**

The HyperLink control is like the HTML <a> element.

Basic syntax for a hyperlink control:

```
<asp:HyperLinkID="HyperLink1"runat="server">
HyperLink
</asp:HyperLink>
```

It has the following important properties:

Property	Description
ImageUrl	Path of the image to be displayed by the control.
NavigateUrl	Target link URL.
Text	The text to be displayed as the link.
Target	The window or frame which loads the linked page.

❖ Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

```
<asp:ImageID="Image1"runat="server">
```

It has the following important properties:

Property	Description
AlternateText	Alternate text to be displayed in absence of the image.
ImageAlign	Alignment options for the control.
ImageUrl	Path of the image to be displayed by the control.

4.7 ASP.NETDataList Server Control

The ASP.NET DataList control is a light weight server side control that works as a container for data items. It is used to display data into a list format to the web pages.

It displays data from the data source. The data source can be either a DataTable or a table from database.

Here, first, we are creating DataList that gets data from a DataTable. This example includes the following files.

ASP.NET DataList Example with DataTable

```

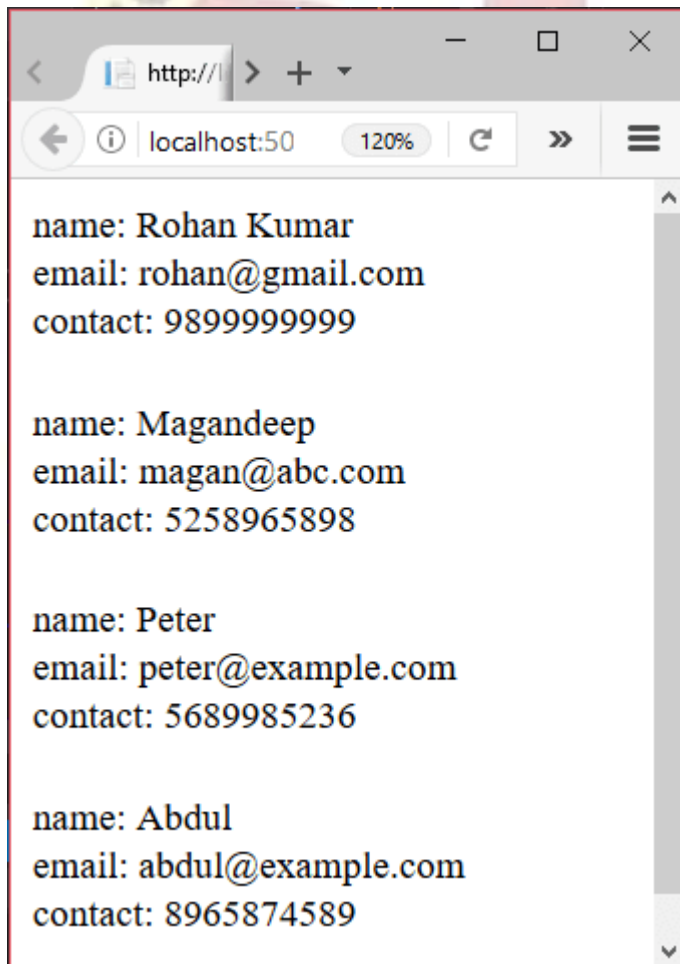
1. <%@ Page Language="C#" AutoEventWireup="true"
2. CodeBehind="DataListExample.aspx.cs" Inherits="AdoNetExample.DataListExam
   ple" %>
3. <!DOCTYPE html>
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6.   <title></title>
7. </head>
8. <body>
9.   <form id="form1" runat="server">
10.    <div>
11.    </div>
12.    <asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSo
       urce1">
13.      <ItemTemplate>
14.        name:
15.        <asp:Label ID="nameLabel" runat="server" Text='<%# Eval("name
           ") %>' />
16.        <br />
17.        email:
18.        <asp:Label ID="emailLabel" runat="server" Text='<%# Eval("email
           ") %>' />
19.        <br />
20.        contact:
21.        <asp:Label ID="contactLabel" runat="server" Text='<%# Eval("con
           tact") %>' />
22.        <br />

```

```
23. <br />
24.     </ItemTemplate>
25. </asp:DataList>
26. <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionStr
    ing="<%$
27.     ConnectionStrings:StudentConnectionString %>"
28.     SelectCommand="SELECT * FROM [student]"></asp:SqlDataSource>
29. </form>
30.</body>
31.</html>
```

Output:

This application produces the following output.



❖ CheckBoxList

The CheckBoxList class is derived from the class System.Web.UI.WebControls.ListControls. ASP.NET CheckBoxList is a web control that can be used to collate the items that can be checked, thus giving the user the ability to select multiple items simultaneously. This list of items in the CheckBoxList can be dynamically generated using the Data Binding functions. The CheckBoxList control class implements different interfaces such as INamingContainer, IPostBackDataHandler, IRepeatInfoUser.

Syntax:

Start Your Free Software Development Course

Web development, programming languages, Software testing & others

The checkboxlist can be created using the design section by dragging and dropping the control from the ASP.NET toolbar window, or else it can also create from the markup section using the following code.

```
<asp: CheckBoxList id="checkboxlist1" AutoPostBack = "True" TextAlign = "Right"
OnSelectedIndexChanged = "CheckList_Clicked" runat="server">
<asp: ListItem> Item 1 </asp: ListItem>
<asp: ListItem> Item 2 </asp: ListItem>
<asp: ListItem> Item 3 </asp: ListItem>
```

The above code will create a checkboxlist named "checkboxlist1" containing items "Item1", "Item2" and "item3" created using the element <asp:ListItem>. These items contain a checkbox preceding every element in the list.

❖ **RadioButtonList Control**

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

The RadioButtonList control supports three important properties that affect its layout:

RepeatColumns: It displays the number of columns of radio buttons.

RepeatDirection: The direction that the radio buttons repeat. By default RepeatDirection value is vertical. Possible values are Horizontal and Vertical.

RepeatLayout: Determines whether the radio buttons display in an HTML table.

Possible values are as follows:

- [
- Table
- Flow
- OrderedList
- UnorderedList

Example

```
using System;
using System.Web.UI.WebControls;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "You have selected </br> Item=" +
            RadioButtonList1.SelectedItem.Text + "</br> Value =" +
            RadioButtonList1.SelectedValue + "</br> Index =" +
            RadioButtonList1.SelectedIndex ;
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
```

```

if (RadioButtonList1.RepeatDirection == RepeatDirection.Vertical)
{
    RadioButtonList1.RepeatDirection = RepeatDirection.Horizontal;
}
else
{
    RadioButtonList1.RepeatDirection = RepeatDirection.Vertical;
}
}
}

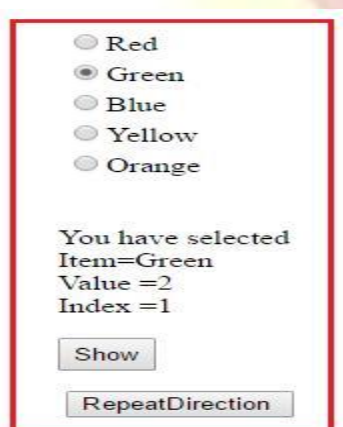
```

You can add items in RadioButtonList through item collection using property window.

```

<asp:RadioButtonList ID="RadioButtonList1" runat="server">
    <asp:ListItem Value="1">Red</asp:ListItem>
    <asp:ListItem Value="2">Green</asp:ListItem>
    <asp:ListItem Value="3">Blue</asp:ListItem>
    <asp:ListItem Value="4">Yellow</asp:ListItem>
    <asp:ListItem Value="5">Orange</asp:ListItem>
</asp:RadioButtonList>

```



Red
 Green
 Blue
 Yellow
 Orange

You have selected
 Item=Green
 Value =2
 Index =1

❖ **ASP.NET DropDown**❖ **List**

The DropDownList is a web server control which is used to create an HTML Select component. It allows us to select an option from the dropdown list. It can contain any number of items

ASP.NET provides a tag to create DropDownList for web application. The following is the Syntax of DropDownList tag.

- `<asp:DropDownList id="DropDownList1" runat="server"`
- `DataSource="<% databindingexpression %>"`
- `DataTextField="DataSourceField"`
- `DataValueField="DataSourceField"`
- `AutoPostBack="True|False"`
- `OnSelectedIndexChanged="OnSelectedIndexChangedMethod">`
- `<asp:ListItem value="value" selected="True|False">`
- `Text`
- `</asp:ListItem>`
- `</asp:DropDownList>`

❖ **LIST BOX**

The ListBox represents a Windows control to display a list of items to a user. A user can select an item from the list. It allows the programmer to add items at design time by using the properties window or at the runtime.

A list box is a graphical control element that allows the user to select one or more items from a list contained within a static, multiple line text box. The user clicks inside the box on an item to select it, sometimes in combination with the \hat{u} Shift or Ctrl in order to make multiple selections.

❖ ASP.NET DataGrid

.NET Framework provides DataGrid control to display data on the web page. It was introduced in .NET 1.0 and now has been deprecated. DataGrid is used to display data in scrollable grid. It requires data source to populate data in the grid.

It is a server side control and can be dragged from the toolbox to the web form. Data Source for the DataGrid can be either a DataTable or a database. Let's see an example, how can we create a DataGrid in our application.

Repeater Controls in ASP.NET

The Repeater control is used to display a repeated list of items that are bound to the control. The Repeater control may be bound to a database table, an XML file, or another list of items.

Repeater is a Data Bind Control. Data Bind Controls are container controls. Data Binding is the process of creating a link between the data source and the presentation UI to display the data.

ASP .Net provides rich and wide variety of controls, which can be bound to the data.

Repeater has 5 inline template to format it:

- <HeaderTemplate>
- <FooterTemplate>
- <ItemTemplate>
- <AlternatingItemTemplate>
- <SeperatorTemplate>
- <AlternatingItemTemplate>

HeaderTemplate: This template is used for elements that you want to render once before your ItemTemplate section.

FooterTemplate: - This template is used for elements that you want to render once after your ItemTemplate section.

ItemTemplate: This template is used for elements that are rendered once per row of data. It is used to display records

AlternatingItemTemplate: This template is used for elements that are rendered every second row of data. This allows you to alternate background colors. It works on even number of records only.

SeperatorTemplate: It is used for elements to render between each row, such as line breaks.

Some point about Repeater Control

- It is used to display backend result set. It is used to display multiple tuple.
- It is an unformatted control. The Repeater control is a basic templated data-bound list. It has no built-in layout or styles, so you must explicitly declare all layout, formatting, and style tags within the control's templates.
- The Repeater control is the only Web control that allows you to split markup tags across the templates. To create a table using templates, include the begin table tag (<table>) in the HeaderTemplate, a single table row tag (<tr>) in the ItemTemplate, and the end table tag (</table>) in the FooterTemplate.
- The Repeater control has no built-in selection capabilities or editing support. You can use the ItemCommand event to process control events that are raised from the templates to the control.

UNIT-5

5.1 Request and Response object

Request Object

The request object is an instance of the System.Web.HttpRequest class. It represents the values and properties of the HTTP request that makes the page loading into the browser.

The information presented by this object is wrapped by the higher level abstractions (the web control model). However, this object helps in checking some information such as the client browser and cookies.

Properties and Methods of the Request Object

The following table provides some noteworthy properties of the Request object:

Property	Description
AcceptTypes	Gets a string array of client-supported MIME accept types.
ApplicationPath	Gets the ASP.NET application's virtual application root path on the server.
Browser	Gets or sets information about the requesting client's browser capabilities.
ContentEncoding	Gets or sets the character set of the entity-body.
ContentLength	Specifies the length, in bytes, of content sent by the client.
ContentType	Gets or sets the MIME content type of the incoming request.
Cookies	Gets a collection of cookies sent by the client.
FilePath	Gets the virtual path of the current request.
Files	Gets the collection of files uploaded by the client, in multipart MIME format.
Form	Gets a collection of form variables.
Headers	Gets a collection of HTTP headers.
HttpMethod	Gets the HTTP data transfer method (such as GET, POST, or HEAD) used by the client.
InputStream	Gets the contents of the incoming HTTP entity body.
IsSecureConnection	Gets a value indicating whether the HTTP connection uses

	secure sockets (that is, HTTPS).
QueryString	Gets the collection of HTTP query string variables.
RawUrl	Gets the raw URL of the current request.
RequestType	Gets or sets the HTTP data transfer method (GET or POST) used by the client.
ServerVariables	Gets a collection of Web server variables.
TotalBytes	Gets the number of bytes in the current input stream.
Url	Gets information about the URL of the current request.
UrlReferrer	Gets information about the URL of the client's previous request that is linked to the current URL.
UserAgent	Gets the raw user agent string of the client browser.
UserHostAddress	Gets the IP host address of the remote client.
UserHostName	Gets the DNS name of the remote client.
UserLanguages	Gets a sorted string array of client language preferences.

The following table provides a list of some important methods:

Method	Description
BinaryRead	Performs a binary read of a specified number of bytes from the current input stream.

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from object.)
GetType	Gets the Type of the current instance.
MapImageCoordinates	Maps an incoming image-field form parameter to appropriate x-coordinate and y-coordinate values.
MapPath(String)	Maps the specified virtual path to a physical path.
SaveAs	Saves an HTTP request to disk.
ToString	Returns a String that represents the current object.
ValidateInput	Causes validation to occur for the collections accessed through the Cookies, Form, and QueryString properties.

Response Object

The Response object represents the server's response to the client request. It is an instance of the System.Web.HttpResponse class.

In ASP.NET, the response object does not play any vital role in sending HTML text to the client, because the server-side controls have nested, object oriented methods for rendering themselves.

However, the HttpResponse object still provides some important functionalities, like the cookie feature and the Redirect() method. The Response.Redirect() method allows transferring the user to another page, inside as well as outside the application. It requires a round trip.

Properties and Methods of the Response Object

The following table provides some noteworthy properties of the Response object:

Property	Description
Buffer	Gets or sets a value indicating whether to buffer the output and send it after the complete response is finished processing.
BufferOutput	Gets or sets a value indicating whether to buffer the output and send it after the complete page is finished processing.
Charset	Gets or sets the HTTP character set of the output stream.
ContentEncoding	Gets or sets the HTTP character set of the output stream.
ContentType	Gets or sets the HTTP MIME type of the output stream.
Cookies	Gets the response cookie collection.
Expires	Gets or sets the number of minutes before a page cached on a browser expires.
ExpiresAbsolute	Gets or sets the absolute date and time at which to remove cached information from the cache.
HeaderEncoding	Gets or sets an encoding object that represents the encoding for the current header output stream.
Headers	Gets the collection of response headers.
IsClientConnected	Gets a value indicating whether the client is still connected to the server.
Output	Enables output of text to the outgoing HTTP response stream.

OutputStream	Enables binary output to the outgoing HTTP content body.
RedirectLocation	Gets or sets the value of the Http Location header.
Status	Sets the status line that is returned to the client.
StatusCode	Gets or sets the HTTP status code of the output returned to the client.
StatusDescription	Gets or sets the HTTP status string of the output returned to the client.
SubStatusCode	Gets or sets a value qualifying the status code of the response.
SuppressContent	Gets or sets a value indicating whether to send HTTP content to the client.

The following table provides a list of some important methods:

Method	Description
AddHeader	Adds an HTTP header to the output stream. AddHeader is provided for compatibility with earlier versions of ASP.
AppendCookie	Infrastructure adds an HTTP cookie to the intrinsic cookie collection.
AppendHeader	Adds an HTTP header to the output stream.
AppendToLog	Adds custom log information to the InterNET Information Services (IIS) log file.

BinaryWrite	Writes a string of binary characters to the HTTP output stream.
ClearContent	Clears all content output from the buffer stream.
Close	Closes the socket connection to a client.
End	Sends all currently buffered output to the client, stops execution of the page, and raises the EndRequest event.
Equals(Object)	Determines whether the specified object is equal to the current object.
Flush	Sends all currently buffered output to the client.
GetType	Gets the Type of the current instance.
Pics	Appends a HTTP PICS-Label header to the output stream.
Redirect(String)	Redirects a request to a new URL and specifies the new URL.
Redirect(String, Boolean)	Redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.
SetCookie	Updates an existing cookie in the cookie collection.
ToString	Returns a String that represents the current

	Object.
TransmitFile(String)	Writes the specified file directly to an HTTP response output stream, without buffering it in memory.
Write(Char)	Writes a character to an HTTP response output stream.
Write(Object)	Writes an object to an HTTP response stream.
Write(String)	Writes a string to an HTTP response output stream.
WriteFile(String)	Writes the contents of the specified file directly to an HTTP response output stream as a file block.
WriteFile(String, Boolean)	Writes the contents of the specified file directly to an HTTP response output stream as a memory block.

Example

The following simple example has a text box control where the user can enter name, a button to send the information to the server, and a label control to display the URL of the client computer.

The content file:

```
<% @PageLanguage="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="server_side._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```

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

<headrunat="server">

<title>Untitled Page</title>

</head>

<body>

<formid="form1"runat="server">

<div>

    Enter your name:

<br/>

<asp:TextBoxID="TextBox1"runat="server"></asp:TextBox>

<asp:ButtonID="Button1"runat="server"OnClick="Button1_Click"Text="Submit"/>

<br/>

<asp:LabelID="Label1"runat="server"/>

</div>

</form>

</body>

</html>

```

The code behind Button1_Click:

```

protectedvoidButton1_Click(object sender,EventArgs e){

if(!String.IsNullOrEmpty(TextBox1.Text)){

// Access the HttpServerUtility methods through

```

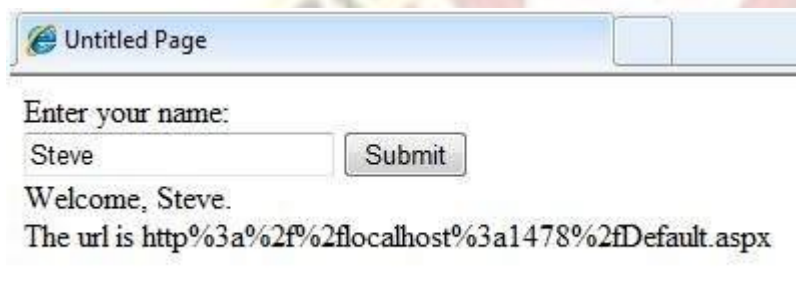
```
// the intrinsic Server object.

Label1.Text="Welcome, "+Server.HtmlEncode(TextBox1.Text)+" . <br/> The url is
"+Server.UrlEncode(Request.Url.ToString())

}

}
```

Run the page to see the following result:



5.2 COOKIES

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

To Create a Cookie

The "Response.Cookies" command is used to create cookies.

Note: The Response.Cookies command must appear BEFORE the <html> tag.

In the example below, we will create a cookie named "firstname" and assign the value "Alex" to it:

```
<%
Response.Cookies("firstname")="Alex"
%>
```

It is also possible to assign properties to a cookie, like setting a date when the cookie should expire:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May 10,2012#
%>
```

5.3 OLEDB CONNECTION CLASS

An OleDbConnection object supports a connection to an OLE DB data provider. In practice, you usually use OLE DB connections with all data providers except Microsoft's SQL Server. Note that, depending on the OLE DB data provider, not all properties of an OleDbConnection object may be supported.

A central property of connection objects is the ConnectionString property, which holds a string full of attribute/value pairs that contain data needed to log on to a data provider and choose a specific database. These attribute/value pairs are specific to the data provider you're using, and make up a list of items separated by semicolons. You can either assign a connection string to the connection's ConnectionString property, or you can pass the connection string to the connection object's constructor, like this:

```
Dim ConnectionString As String = "Provider=SQLOLEDB.1;Integrated " & _
Security=SSPI;Persist Security Info=False;Initial " & _
"Catalog=pubs;Packet Size=4096;Workstation ID=STEVE;" & _
"Use Encryption for Data=False"
```

```
Dim Connection1 As OleDbConnection = New OleDbConnection(ConnectionString)
```

If you have no idea what a connection string should look like for a specific data provider and database, use the visual tools built into Visual Basic to construct a few sample strings to that

data provider, which you can either use directly in code or modify as you need. To do that, create a connection to the source you want to use, then drag a data adapter to a project's main form, which creates both data connection and data adapter objects. Then take a look at the connection object's `ConnectionString` property in the Properties window.

Tip The most common attribute/value pairs used in OLE DB connection strings are also supported with properties of connection objects, such as `DataSource`, `Database`, `UserId`, and `Password`, which means that when you work with a connection object, you can either set the `ConnectionString` property as a string, or you can set various connection properties one-by-one and let Visual Basic create the connection string for you (unless your OLE DB provider requires data not supported by the connection object's properties).

After you've created a connection object, you can open it with the `Open` method, and assign it to the `Connection` property of a command object. (To specify the SQL you want to use, you can pass that SQL to the command object's constructor.) Then you can use the command object with a data adapter. For example, you might assign the command object to the `SelectCommand` property of a data adapter, and you can use the data adapter's `Fill` method to execute that command and fill a dataset. When done with the connection, use its `Close` method to close it. (The connection won't be closed otherwise, even if the connection object goes out of scope.)

Tip If your application uses a number of connections, you should use connection pooling to improve performance. (Connection pooling lets you keep a cache of connections without having to create new ones all the time.) When you use the OLE DB .NET data provider, connection pooling is enabled automatically.

5.4 Command class

The `Command` class is provided by all standard ADO.NET providers, and it almost always encapsulates a SQL statement or a stored procedure call that can be executed against a data source. Command objects can retrieve rows; directly insert, delete, or modify records; calculate totals and averages; alter the structure of a database; or fill a disconnected `DataSet` when used with a `DataAdapter`.

At a bare minimum, every Command must reference a Connection object, which it uses to communicate with the data source and define a few key properties, such as CommandText (the stored procedure or embedded SQL command) and CommandType. The Command class is provided in several provider-specific varieties, including SqlCommand and OleDbCommand.

To execute a Command, you use one of the Command object methods, including ExecuteNonQuery(), ExecuteReader(), and ExecuteScalar(), depending on the type of Command. Occasionally, a provider may define an additional method, such as the ExecuteXmlReader() method offered by the SQL Server provider, which retrieves query results as an XML document. Other than this discrepancy, the Command objects provided by the core set of ADO.NET providers are virtually identical.

5.5 Transaction Class

The **Transaction** class contains methods used by developers implementing resource managers for enlistment. It also provides functionalities for cloning a **transaction** and controlling the current **transaction** context. You can obtain the current **transaction**, if one is set, using the static Current property.

5.6 Data Adapter

The DataAdapter works as a bridge between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

DataAdapter Class Signature

DataAdapter Constructors

Constructors	Description
DataAdapter()	It is used to initialize a new instance of a DataAdapter class.
DataAdapter(DataAdapter)	It is used to initializes a new instance of a DataAdapter class from an existing object of the same type.

5.7 Data set class

It is a collection of data tables that contain the data. It is used to fetch data without interacting with a Data Source that's why, it also known as disconnected data access method. It is an in-memory data store that can hold more than one table at the same time. We can use DataRelation object to relate these tables. The DataSet can also be used to read and write data as XML document.

ADO.NET provides a DataSet class that can be used to create DataSet object. It contains constructors and methods to perform data related operations.

DataSet Class Signature

1. public class DataSet : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource,
2. System.ComponentModel.ISupportInitializeNotification, System.Runtime.Serialization.ISerializable,
3. System.Xml.Serialization.IXmlSerializable

5.8 ADVANCED ISSUES

Email:

Email is a computer based method of sending messages from one computer user to another. These messages usually consist of individual pieces of text which you can send to another computer user even if the other user is not logged in (i.e. using the computer) at the time you send your message. The message can then be read at a later time. This procedure is analogous to sending and receiving a letter.

When mail is received on a computer system, it is usually stored in an electronic mailbox for the recipient to read later. Electronic mailboxes are usually special files on a computer which can be accessed using various commands. Each user normally has their individual mailbox.

Host-based mail systems

The original email systems allowed communication only between users who logged into the same host or "mainframe". This could be hundreds or even thousands of users within an organization.

By 1966 (or earlier, it is possible that the SAGE system had something similar some time before), such systems allowed email between different organizations, so long as they ran compatible operating systems.

Examples include BITNET, IBM PROFS, Digital Equipment Corporation ALL-IN-1 and the original Unix mail.

LAN-based mail systems

From the early 1980s, networked personal computers on LANs became increasingly important. Server-based systems similar to the earlier mainframe systems were developed. Again these systems initially allowed communication only between users logged into the same server infrastructure. Eventually these systems could also be linked between different organizations, as long as they ran the same email system and proprietary protocol.

Examples include cc:Mail, Lantastic, WordPerfect Office, Microsoft Mail, Banyan VINES and Lotus Notes - with various vendors supplying gateway software to link these incompatible systems.

Early interoperability among independent systems included:uucp was used as an open "glue" between differing mail systems, primarily over dialup telephones

- ARPANET which was the forerunner of today's Internet
- CSNet which used dial-up telephone access to link additional sites to the ARPANET and then Internet

5.9 Working with IIS:

IIS (Internet Information Server) is a group of Internet servers (including a Web or Hypertext Transfer Protocol server and a File Transfer Protocol server) with additional capabilities for Microsoft's Windows NT and Windows 2000 Server operating systems. IIS is Microsoft's entry to compete in the Internet server market that is also addressed by Apache, Sun Microsystems, O'Reilly, and others. With IIS, Microsoft includes a set of programs for building and administering Web sites, a search engine, and support for writing Web-based applications that access databases. Microsoft points out that IIS is tightly integrated with the Windows NT and 2000 Servers in a number of ways, resulting in faster Web page serving.

A typical company that buys IIS can create pages for Web sites using Microsoft's Front Page product (with its WYSIWYG user interface). Web developers can use Microsoft's Active Server Page (ASP) technology, which means that applications - including ActiveX controls - can be

imbedded in Web pages that modify the content sent back to users. Developers can also write programs that filter requests and get the correct Web pages for different users by using Microsoft's Internet Server Application Program Interface (ISAPI) interface. ASPs and ISAPI programs run more efficiently than common gateway interface (CGI) and server-side include (SSI) programs, two current technologies. (However, there are comparable interfaces on other platforms.)

Microsoft includes special capabilities for server administrators designed to appeal to Internet service providers (ISPs). It includes a single window (or "console") from which all services and users can be administered. It's designed to be easy to add components as snap-ins that you didn't initially install. The administrative windows can be customized for access by individual customers.

Worker process isolation mode Provides an easy way to insulate Web applications from each other, so that problems with one Web application don't impact the other Web applications on Microsoft Internet Information Services(IIS). IIS 6.0 allows you to organize applications into application pools. Each application pool is a completely independent entity, served by one or more worker processes. Usually, a Windows administrator will create a separate application pool for each Web application that the server hosts -- but a single application pool can host multiple applications. Of course, this raises the question of how application pools can isolate IIS Web applications from each other. True isolation is possible because Windows differentiates between code that is running in kernel mode vs. code that is running in user mode.

Windows runs core IIS components, such as HTTP.SYS and the WWW service, in kernel mode. Each application pool contains its own kernel-mode queue. This means that HTTP.SYS is able to route inbound requests directly to a queue that is dedicated to a specific application pool, all within kernel mode. Application pools are separated from each other by process boundaries.

Worker processes are dedicated to a specific application pool to actually service requests. If a failure occurs, it usually happens within a worker process. However, since worker processes are bound to particular application pools, a worker process failure will only affect the application in which it resides, but no others. The really cool part is that IIS provides mechanisms for monitoring the health of a worker process. If a worker process fails, the process can be restarted without the end user even being aware of the failure.

5.10 The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

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

The attributes of the Page directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page_Load.
Buffer	The Boolean value that enables or disables HTTP response buffering.
ClassName	The class name for the page.
ClientTarget	The browser for which the server controls should render content.
CodeFile	The name of the code behind file.
Debug	The Boolean value that enables or disables compilation with debug symbols.
Description	The text description of th

5.11 Error Handling

Although ASP.NET can detect all runtime errors, still some subtle errors may still be there. Observing the errors by tracing is meant for the developers, not for the users.

Hence, to intercept such occurrence, you can add error handling settings in the web.config file of the application. It is application-wide error handling. For example, you can add the following lines in the web.config file:

```
<configuration>
<system.web>
<customErrorsmode="RemoteOnly"defaultRedirect="GenericErrorPage.htm">
<errorstatusCode="403"redirect="NoAccess.htm" />
<errorstatusCode="404"redirect="FileNotFound.htm"/>
</customErrors>
</system.web>
</configuration>
```

The <customErrors> section has the possible attributes:

- **Mode** : It enables or disables custom error pages. It has the three possible values:
 - **On** : displays the custom pages.
 - **Off** : displays ASP.NET error pages (yellow pages)
 - **remoteOnly** : It displays custom errors to client, display ASP.NET errors locally.
- **defaultRedirect** : It contains the URL of the page to be displayed in case of unhandled errors.

To put different custom error pages for different type of errors, the <error> sub tags are used, where different error pages are specified, based on the status code of the errors.

To implement page level error handling, the Page directive could be modified:

```
<% @PageLanguage="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
```

```
Inherits="errorhandling._Default"Trace="true"ErrorPage="PageError.htm" %>
```

Because ASP.NET Debugging is an important subject in itself, so we would discuss it in the next chapter separately.

5.12 SECURITY Authentication

Authentication is used by a server when the server needs to know exactly who is accessing their **information** or **site**. **Authentication** is used by a client when the client needs to know that the server is system it claims to be. **Inauthentication**, the user or computer has to prove its identity to the server or client.

5.13 IP Address

IP stands for internet protocol. It is a protocol defined in the TCP/IP model used for sending the packets from source to destination. The main task of IP is to deliver the packets from source to the destination based on the IP addresses available in the packet headers. IP defines the packet structure that hides the data which is to be delivered as well as the addressing method that labels the datagram with a source and destination information.

An IP protocol provides the connectionless service, which is accompanied by two transport protocols, i.e., TCP/IP and UDP/IP, so internet protocol is also known as TCP/IP or UDP/IP.

The first version of IP (Internet Protocol) was IPv4. After IPv4, IPv6 came into the market, which has been increasingly used on the public internet since 2006.

5.14 Secure by SSL& CLIENT certificates

Server Certificate

Server certificates (SSL certificates) are used to authenticate the identity of a server. When installed on a website, an SSL certificate turns the protocol on the website from HTTP to HTTPS [Difference b/w HTTP and https] and installs indicators that vouch for the authenticity of the website. Thus, users can know the website belongs to the said entity. Apart from authentication, SSL certificates also facilitate Encryption. Meaning, any information a user sends to the server is protected from the reaches of any ill-intended 3rd party.

Client Certificate

Contrary to Server certificates (SSL certificates), Client certificates are used to validate the identity of a client (user). The user, in this case, might be a website user or an email user. Simply put, it works as a password, but without any intervention/input from the user. This way, the server makes sure that it's connecting to the permitted user and that party is safe to communicate with.

Now you might be wondering 'Don't passwords do the same thing?' Well, sometimes passwords are not good enough. We often fall prey to password cracking techniques such as brute-force attacks and keyloggers. That's why passwords are no longer sufficient when you have some really highly-sensitive information at stake.

So, there might be some documents or files that you want only designated people to access. But as passwords are not secure enough, you'll have to explore your options. That's where Client certificates come in. Instead of validating people via passwords, Client certificates authenticate people by the systems they use. If the user doesn't have the granted permissions, he/she won't be granted access. To make it even more secure, you can combine the use of client certificates with passwords. In technical terms, this is called 'Two-factor Authentication.' It is an absolute must for organizations dealing with sensitive data –both internal as well as external. And you know what happens when you don't employ two-factor authentication? Just ask Equifax!

Client certificates also use public key infrastructure (PKI) for authentication, just like Server certificates. However, there is one significant difference between the two. Unlike Server certificates, Client certificates don't encrypt any data; they're installed for validation purposes only.

Client Certificate vs Server certificate: What's the difference?

Server Certificate	Client Certificates
Server certificates are used to authenticate server identity to the client(s).	Client certificates are used to authenticate the client (user) identity to the server.
Server certificates encrypt data-in-transit.	No encryption of data takes place in case of Client certificates.
Server Certificates are based on PKI.	Client certificates are based on PKI.
Example: SSL certificates	Example: E-mail Client certificates