WEB TECHNOLOGIES

UNIT – 1

Internet Basics: Basic Concepts – Internet Domains – IP Address – TCP/IP Protocol – The WWW – The Telnet –– Introduction to HTML: Web server - Web client / browser - Tags – Text Formatting – Lists – Tables – Linking Documents - Frames.

INTERNET BASICS

Network of network is called Internet. The Internet is a global collection of computer networks that are linked together by devices called routers and use a common set of protocols for data transmission known as TCP/IP (transmission control protocol / Internet protocol). The primary purpose of the Internet is to facilitate the sharing of information.

BROWSER

A WWW browser is software on your computer that allows you to access the World Wide Web. Examples include Netscape Navigator and Microsoft Internet Explorer.

HYPERTEXT

Hypertext is text that contains electronic links to other text. In other words, if you click on hypertext it will take you to other related material.

URL (UNIFORM RESOURCE LOCATOR)

Links between documents are achieved by using an addressing scheme. That is, in order to link to another document or item (sound, picture, movie), it must have an address. That address is called its URL. The URL identifies the host computer name, directory path, and file name of the item.

HTTP (HYPER TEXT TRANSFER PROTOCOL)

HTTP is the protocol used to transfer hypertext or hypermedia documents.

WEBSITE

Collection of Web pages is called Website. Example : www.google.com

WEBSERVER

Every Website sits on a computer known as a Web server. This server is always connected to the internet. Every Web server that is connected to the Internet is given a unique address made up of a series of four numbers between 0 and 256 separated by periods. For example, 68.178.157.132 or 68.122.35.127.

HOMEPAGE

A home page is usually the starting point for locating information at a WWW site.

ISP (INTERNET SERVICE PROVIDER)

ISP stands for Internet Service Provider. They are the companies who provide you service in terms of internet connection to connect to the internet. You will buy space on a Web Server from any Internet Service Provider. This space will be used to host your Website.

CLIENTS AND SERVER

If a computer has a web browser installed, it is known as a client. A host computer that is capable of providing information to others is called a server. A server requires special software in order to provide web documents to others.

INTERNET DOMAIN

The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network.

The Internet Assigned Numbers Authority (IANA) today distinguishes the following groups of top-level domains:

- generic top-level domains (gTLD): Top-level domains with three or more characters
  - .com – Commercial purpose – www.google.com
  - .org – Organization – www.tnpsc.org
  - .edu – Education – www.annauniv.edu
  - .net – Network
  - .gov – Government – www.tn.gov.nic.in
  - .mil – US Military
- country-code top-level domains (ccTLD): Two letter domains established for countries or territories
  - .in – India
  - .af – Afghanistan
  - .au – Australia
  - .bd – Bangladesh
  - .us – USA
  - .jp – Japan

IP ADDRESS

An IP address (abbreviation of Internet Protocol address) is an identifier assigned to each computer and other device (e.g., printer, router, mobile device, etc.) connected to a TCP/IP network[1] that is used to locate and identify the node in communications with other nodes on the network.

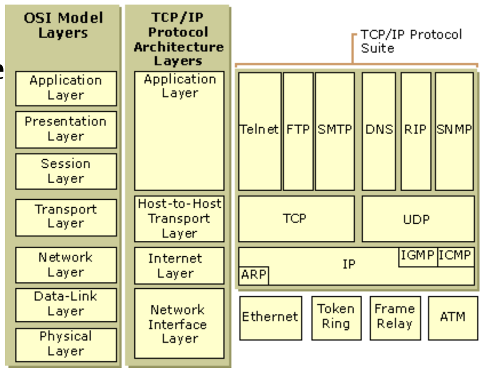
Two Types of IP Address

- Version 4 of the Internet Protocol (IPv4) defines an IP address as a 32-bit number.
- IP (IPv6), using 128 bits for the IP address, was developed in 1995

TCP / IP PROTOCOL

TCP/IP protocols map to a four-layer conceptual model known as the DARPA model , named after the U.S. government agency that initially developed TCP/IP. The four layers of the DARPA model are: Application, Transport, Internet, and Network Interface. Each layer in the DARPA model corresponds to one or more layers of the seven-layer Open Systems Interconnection (OSI) model.

Figure 1.1 TCP/IP Protocol Architecture

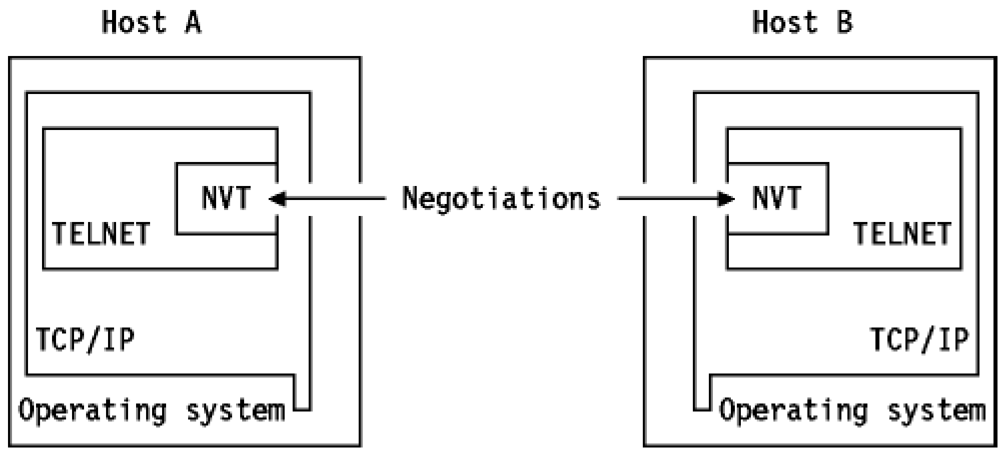


THE TELNET

- TELNET is a general protocol, meant to support logging in from almost any type of terminal to almost any type of computer.
- It allows a user at one site to establish a TCP connection to a login server or terminal server at another site.
- A TELNET server generally listens on TCP Port 23.

How it works

- A user is logged in to the local system, and invokes a TELNET program (the TELNET client) by typing telnet xxx.xxx.xxx
  where xxx.xxx.xxx is either a host name or an IP address.



NVT – NETWORK VIRTUAL TERMINAL

INTRODUCTION TO HTML

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

A Simple HTML Document

Example

```
<!DOCTYPE html>
<html>
<head>
<title>PageTitle</title>
</head>
```

```
<body>
<h1>MyFirstHeading</h1>
<p>Myfirstparagraph.</p>
</body>
</html>
```

Example Explained

- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The <p> element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

```
<tagname>content goes here...</tagname>
```

- HTML tags normally come in pairs like <p> and </p>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Tip: The start tag is also called the opening tag, and the end tag the closing tag.

HTML Formatting Elements

In the previous chapter, you learned about the HTML **style attribute**.
HTML also defines special **elements** for defining text with a special **meaning**.
HTML uses elements like <b> and <i> for formatting output, like bold or italic text.

Formatting elements were designed to display special types of text:

- <b> - Bold text
- <strong> - Important text
- <i> - Italic text
- <em> - Emphasized text
- <mark> - Marked text
- <small> - Small text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

Example

```
<!DOCTYPE html>
<html>
<body>
<p>This text is normal.</p>
<p><b><i>This text is bold.</i></b></p>
</body>
</html>
```

HTML List Example

An Unordered List:

- Item
- Item
- Item
- Item

An Ordered List:

1. First item
2. Second item
3. Third item
4. Fourth item

Unordered HTML List

An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag. The list items will be marked with bullets (small black circles) by default:

```
<!DOCTYPE html>
<html>
<body>
<h2>An unordered HTML list</h2>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
</body>
</html>
```

Unordered HTML List - Choose List Item Marker

The CSS **list-style-type** property is used to define the style of the list item marker:

Value	**Description**
Disc	Sets the list item marker to a bullet (default)
Circle	Sets the list item marker to a circle
Square	Sets the list item marker to a square
None	The list items will not be marked

```
<!DOCTYPE html>
<html>
<body>
<h2>Unordered List with Square Bullets</h2>
<ul style="list-style-type:square">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
```

```
</ul>
</body>
</html>
```

Ordered HTML List

An ordered list starts with the **<ol>** tag. Each list item starts with the **<li>** tag.

Ordered HTML List - The Type Attribute

The **type** attribute of the <ol> tag, defines the type of the list item marker:

Type	**Description**
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Example

```
<!DOCTYPE html>
<html>
<body>
<h2>Ordered List with Numbers</h2>
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
</body>
</html>
```

Defining an HTML Table

An HTML table is defined with the **<table>** tag.

Each table row is defined with the **<tr>** tag. A table header is defined with the **<th>** tag. By default, table headings are bold and centered. A table data/cell is defined with the **<td>** tag.

```
<!DOCTYPE html>
<html>
<body>
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>80</td>
  </tr>
</table>
</body> </html>
```

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

In HTML, links are defined with the **<a>** tag:

<a href="*url*">*link text*</a>

Example

```
<!DOCTYPE html>
<html>
<body>
<p><a href="html_images.asp">HTML Images</a> is a link to a page on this website.</p>1
<p><a href="http://www.w3.org/">W3C</a> is a link to a website on the World Wide
Web.</p>
</body>
</html>
```

The **href** attribute specifies the destination address of the link.

The **link text** is the visible part (Visit our HTML tutorial).

HTML FRAMES

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages:

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back button* might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Example

Let's put above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
<frameset cols="25%,50%,25%">
  <frame name="left" src="/html/top_frame.htm" />
  <frame name="center" src="/html/main_frame.htm" />
  <frame name="right" src="/html/bottom_frame.htm" />
  <noframes>
  <body>
    Your browser does not support frames.
  </body>
  </noframes>
</frameset>
</html>
```

UNIT II

JavaScript: JavaScript in Web Pages – The Advantages of JavaScript – Writing JavaScript into HTML – Syntax – Operators and Expressions – Constructs and conditional checking – Functions – Placing text in a browser – Dialog Boxes – Form object's methods – Built in objects – user defined objects.

JAVASCRIPT IN WEB PAGES

JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

This tutorial is about JavaScript, and how JavaScript works with HTML and CSS.

Did You Know?

JavaScript and Java are completely different languages, both in concept and design.

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997. ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- 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.

ADVANTAGES OF JAVASCRIPT

- ➢ Interpreted languages
- ➢ Embedded within HTML
- ➢ Minimal Syntax – Easy to Learn
- ➢ Quick Development
- ➢ Designed for Simple, Small Programs
- ➢ Performance
- ➢ Procedural Capabilities
- ➢ Easy Debugging and Testing
- ➢ Platform Independence / Architectural Neutral

SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the HTML tags in a web page.

You can place the The script tag takes two important attributes:

- **Language:** This attribute specifies what scripting language you are using.• Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type:** This attribute is what is now recommended to indicate the scripting• language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

FIRST JAVASCRIPT CODE

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
   document.write ("Hello World!")
//-->
</script>
</body>
</html>
```

This code will produce the following result

```
Hello World!
```

Case Sensitivity

Java Script is a Case Sensitivity Languages.

Comments in JavaScript

- JavaScript supports both C-style and C++-style comments. Thus:
- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines. JavaScript also recognizes the HTML comment opening sequence <!--. Java Script treats this as a Single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

JAVASCRIPT VARIABLES

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it.

Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

Variable Initialization can be done as follows

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

OPERATORS

Java Script supports the following types of Operators

- ➢ Arithmetic Operators
- ➢ Assignment Operators
- ➢ Logical (or Relational) Operators
- ➢ Comparison Operators
- ➢ Conditional (or ternary) Operators

Let’s have a look at all the operators one by one.

ARITHMETIC OPERATORS

Arithmetic operators are used to perform arithmetic on numbers:

Operator	**Description**	**Example**
Remainder (%)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
Increment (++)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ returns 3 and, only then, sets xto 4.
Decrement (--)	Unary operator. Subtracts one from its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x-- returns 3 and, only then, sets xto 2.
Unary negation (-)	Unary operator. Returns the negation of its operand.	If x is 3, then -xreturns -3.
Unary plus (+)	Unary operator. Attempts to convert the operand to a number, if it is not already.	+"3" returns 3. +true returns 1.
Exponentiation operator (**)	Calculates the base to the exponent power, that is, $\text{base}^{\text{exponent}}$	2 ** 3 returns 8. 10 ** -1 returns 0.1.

ASSIGNMENT OPERATOR

Name	Shorthand operator	Meaning
Assignment	x = y	x = y
Addition assignment	x += y	x = x + y
Subtraction assignment	x -= y	x = x – y
Multiplication assignment	x *= y	x = x * y
Division assignment	x /= y	x = x / y
Remainder assignment	x %= y	x = x % y
Exponentiation assignment	x **= y	x = x ** y
Left shift assignment	x <<= y	x = x << y
Right shift assignment	x >>= y	x = x >> y
Unsigned right shift assignment	x >>>= y	x = x >>> y
Bitwise AND assignment	x &= y	x = x & y
Bitwise XOR assignment	x ^= y	x = x ^ y
Bitwise OR assignment	x |= y	x = x | y

LOGICAL OPERATOR

Operator	Usage	Description
Logical AND(&&)	expr1 && expr2	Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.

Operator	**Usage**	**Description**
Logical OR (||)	expr1 || expr2	Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
Logical NOT (!)	!expr	Returns false if its single operand can be converted to true; otherwise, returns true.

COMPARISON OPERATOR

Operator	**Description**	**Examples returning true**
Equal (==)	Returns true if the operands are equal.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal.	var1 != 4 var2 != "3"
Strict equal(===)	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.	var1 !== "3" 3 !== '3'
Greater than(>)	Returns true if the left operand is greater than the right operand.	var2 > var1 "12" > 2
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2 "2" < 12
Less than or	Returns true if the left operand is less than or equal to the	var1 <= var2

Operator	Description	Examples returning true
equal (<=)	right operand.	var2 <= 5

CONDITIONAL (TERNARY) OPERATOR

The conditional operator is the only JavaScript operator that takes three operands. The operator can have one of two values based on a condition. The syntax is:

condition ? *val1* : *val2*

If condition is true, the operator has the value of val1. Otherwise it has the value of val2. You can use the conditional operator anywhere you would use a standard operator.
For example,

```
var status = (age >= 18) ? 'adult' : 'minor';
```

JAVA SCRIPT PROGRAMMING CONSTRUCTS

- Most programming languages support a common set of constructs. Languages only differ in the syntax used for structuring these constructs.
- Java Script provides a complete range of basic programming constructs. While it is not an object oriented programming environment JavaScript is an Object-based language.
- The constructs provided by JavaScript are as follows

Construct / Statement	**Purpose**	**Example**
Assignment	Assigns the value of an expression to a variable	X = y + z
Data declaration	Declares a variable and optionally assigns a value to it.	Var myVar=10

If	Program execution depends upon the value of returned by the condition. If the value returned is True the program executes else the program does not execute.	If(x>y) {
		Z=X;
		}

While	Repeatedly executes a set of statements until a condition becomes false	While (x!=7) {
		X%=n-n
		}

Switch	Selects from a number of alternatives	Switch (val) {
		Case 1:
		// First alternative
		Break;
		Case 2:
		//Second alternative
		Break;
		Default
		// Default action
		}

For	Repeatedly executes a set of statements until a condition becomes false	For(i=0;i<7;++i) {
		Document.write(x[i]);
		}

Do while	Repeatedly executes a set of statements while a condition is true	Do {
		// Statements
		} while (i>0)

Label	Associates a label with a statement	LabelName: Statement.

CONDITIONAL STATEMENTS

A conditional statement is a set of commands that executes if a specified condition is true. JavaScript supports two conditional statements: if...else and switch.

if...else statement

Use the if statement to execute a statement if a logical condition is true. Use the optional elseclause to execute a statement if the condition is false. An if statement looks as follows:

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

Switch statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
   case 0:
      day = "Sunday";
      break;
   case 1:
      day = "Monday";
      break;
   case 2:
      day = "Tuesday";
      break;
   case 3:
      day = "Wednesday";
      break;
   case 4:
      day = "Thursday";
      break;
   case 5:
      day = "Friday";
      break;
   case 6:
      day = "Saturday";
}
```

FUNCTIONS IN JAVA SCRIPT

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
The parentheses may include parameter names separated by commas:

(parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3)
{
   code to be executed
}
```

- Function **parameters** are the **names** listed in the function definition.
- Function **arguments** are the real **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.
- Functions can be declared anywhere within an HTML file. Preferably, functions are created within the <HEAD>…..</HEAD> tags of the HTML file.

FUNCTION DECLARATION

```
Function printName(user) {
Document.write("<HR/> Your Name is <B><I>");
Document.write(user);
Document.write("</B></I><HR?>);
}
```

}

Where, printName is a function, which has a parameter called user. The parameter user can be passed a value at the time of invoking the function.

FUNCTION RETURN

- When JavaScript reaches a **return statement**, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

```
var x = myFunction(4, 3);// Function is called, return value will end up in x
function myFunction(a, b)
{
   return a * b;        // Function returns the product of a and b
}
```

RECURSIVE FUNCTIONS

Recursion – wherein functions call themselves.

Example : Finding a Factorial Number

```
Function factorial(number) {
        If(number>1) {
                Return number * factorial (number-1);
        }
        Else {
                Return number;
        }
}
```

PLACING TEXT IN A BROWSER

Using JavaScript a string can be written to the browser from within an HTML file. The document object in JavaScript has a method for placing text in a browser. This method is called write().

Example

```
        Document.write(“Test”);
```

Example coding

```
<HTML>
<HEAD><TITLE>Outputting Text</TITLE></HEAD>
<BODY><CENTER><BR/><BR/>
<SCRIPT Languages=”Javascript”>
        Document.write(“Welcome to JavaScript”);
</SCRIPT>
</CENTER></BODY>
</HTML>
```

DIALOG BOXES

JavaScript provides the ability to pickup user input or display small amounts of text to the user by using Dialog Boxes. It appears as a separate window in Browser.

Three types of Dialog Boxes

1. Alert Dialog Box
2. The Prompt Dialog Box
3. The Confirm Dialog Box

Alert Dialog Box

An alert box is often used if you want to make sure information comes through to the user.
When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

window.alert("*sometext*");

Example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Alert</h2>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
    alert("I am an alert box!");
}
</script>
</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

window.confirm("*sometext*");

Example

```
if (confirm("Press a button!") == true) {
    txt = "You pressed OK!";
} else {
    txt = "You pressed Cancel!";
}
```

Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultText");
```

Example

```
var person = prompt("Please enter your name", "Harry Potter");
if (person == null || person == "") {
    txt = "User cancelled the prompt.";
} else {
    txt = "Hello " + person + "! How are you today?";
}
```

JAVASCRIPT FORM OBJECT

The JavaScript Form Object is a property of the document object. This corresponds to an HTML input form constructed with the FORM tag. A form can be submitted by calling the JavaScript submit method or clicking the form submit button.

Form Object Properties

- action - This specifies the URL and CGI script file name the form is to be submitted to. It allows reading or changing the ACTION attribute of the HTML FORM tag.
- elements - An array of fields and elements in the form.
- encoding - This is a read or write string. It specifies the encoding method the form data is encoded in before being submitted to the server. It corresponds to the ENCTYPE attribute of the FORM tag. The default is "application/x-www-form-urlencoded". Other encoding includes text/plain or multipart/form-data.
- length - The number of fields in the elements array. I.E. the length of the elements array.
- method - This is a read or write string. It has the value "GET" or "POST".
- name - The form name. Corresponds to the FORM Name attribute.
- target - The name of the frame or window the form submission response is sent to by the server. Corresponds to the FORM TARGET attribute.

Form Objects

Forms have their own objects.

- button - An GUI pushbutton control. Methods are click(), blur(), and focus(). Attributes:
    - name - The name of the button
    - type - The object's type. In this case, "button".
    - value - The string displayed on the button.
- checkbox - An GUI check box control. Methods are click(), blur(), and focus(). Attributes:
    - checked - Indicates whether the checkbox is checked. This is a read or write value.
    - defaultChecked - Indicates whether the checkbox is checked by default. This is a read only value.

    - name - The name of the checkbox.
    - type - Type is "checkbox".
    - value - A read or write string that specifies the value returned when the checkbox is selected.
- FileUpload - This is created with the INPUT type="file". This is the same as the text element with the addition of a directory browser. Methods are blur(), and focus(). Attributes:
    - name - The name of the FileUpload object.
    - type - Type is "file".
    - value - The string entered which is returned when the form is submitted.
- hidden - An object that represents a hidden form field and is used for client/server communications. No methods exist for this object. Attributes:
    - name - The name of the Hidden object.
    - type - Type is "hidden".
    - value - A read or write string that is sent to the server when the form is submitted.
- password - A text field used to send sensitive data to the server. Methods are blur(), focus(), and select(). Attributes:
    - defaultValue - The default value.
    - name - The name of the password object."
    - type - Type is "password".
    - value - A read or write string that is sent to the server when the form is submitted.
- radio - A GUI radio button control. Methods are click(), blur(), and focus(). Attributes:
    - checked - Indicates whether the radio button is checked. This is a read or write value.
    - defaultChecked - Indicates whether the radio button is checked by default. This is a read only value.
    - length - The number of radio buttons in a group.
    - name - The name of the radio button.
    - type - Type is "radio".
    - value - A read or write string that specifies the value returned when the radio button is selected.
- reset - A button object used to reset a form back to default values. Methods are click(), blur(), and focus(). Attributes:

        - name - The name of the reset object.
        - type - Type is "reset".
        - value - The text that appears on the button. By default it is "reset".
- select - A GUI selection list. This is basically a drop down list. Methods are blur(), and focus(). Attributes:
        - length - The number of elements contained in the options array.
        - name - The name of the selection list.
        - options - An array each of which identifies an options that may be selected in the list.
        - selectedIndex - Specifies the current selected option within the select list
        - type - Type is "select".
- submit - A submit button object. Methods are click(), blur(), and focus(). Attributes:
        - name - The name of the submit button.
        - type - Type is "submit".
        - value - The text that will appear on the button.
- text - A GUI text field object. Methods are blur(), focus(), and select(). Attributes:
        - defaultValue - The text default value of the text field.
        - name - The name of the text field.
        - type - Type is "text".
        - value - The text that is entered and appears in the text field. It is sent to the server when the form is submitted.
- textarea - A GUI text area field object. Methods are blur(), focus(), and select(). Attributes:
        - defaultValue - The text default value of the text area field.
        - name - The name of the text area.
        - type - Type is textarea.

value- The text that is entered and appears in the text area field. It is sent to the server when the form is submitted.

Form Object Methods

- reset() - Used to reset the form elements to their default values.
- submit() - Submits the form as though the submit button were pressed by the user.

Events

- onReset
- onSubmit

JAVASCRIPT BUTTON OBJECT

The JavaScript Button object is a property of the form object. Button FORM syntax is:

```
<INPUT TYPE="button" NAME="myButton" VALUE="Press This" onClick="clickFunction">
```

The type may be one of "button", "submit", or "reset". The name is the name you want to call the button. The onclick and associated function is optional. The clickFunction is run when the button is clicked and must exist, usually in the header.

JAVASCRIPT CHECKBOX OBJECT

The JavaScript Checkbox object is a property of the form object. Checkbox FORM syntax is:

```
<INPUT TYPE="checkbox" NAME="Item1" VALUE="1" CHECKED
onClick="clickFunction">
```

Here's how it looks:

The button belongs to the checkBox group of buttons and any combination in that group should be checked. The clickFunction is run when the button is clicked and must exist, usually in the header. The value is the value returned when the button is checked. The option "CHECKED" sets the button so it is selected when it is initially displayed. Both the onClick and CHECKED parameters are optional. Type, name, and value are required.

JAVASCRIPT FILEUPLOAD OBJECT

The JavaScript FileUpload Object is a property of the form object which allows users to provide a local file as part of their form input thereby allowing for indirect file upload. It can be implement using a form as follows:

```
<FORM>
<INPUT TYPE="file" NAME="elementName">
</FORM>
```

JAVASCRIPT OPTION OBJECT

The JavaScript Option object is a property of the form object. It is one of several options in a selection box. It appears inside a select box as follows:

```
<SELECT NAME="state" SIZE=0>
        <OPTION VALUE="0">
        <OPTION VALUE="1">AL
        <OPTION VALUE="2">AK
        <OPTION VALUE="3">AR
</SELECT>
```

It looks like:

JAVASCRIPT RADIO OBJECT

The JavaScript Radio object is a property of the form object. Radio button FORM syntax is:

```
<FORM>
<INPUT TYPE="radio" NAME="timeofDay" VALUE="Morning" CHECKED
onClick="clickFunction">Morning
<INPUT TYPE="radio" NAME="timeofDay" VALUE="Afternoon"
onClick="clickFunction">Afternoon
<INPUT TYPE="radio" NAME="timeofDay" VALUE="Evening"
```

onClick="clickFunction">Evening

</FORM>

Here's how it looks:

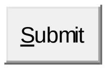
● Morning ○ Afternoon ○ Evening

JAVASCRIPT SUBMIT OBJECT

The JavaScript Submit Object is a property of the form object. Example code:

<INPUT TYPE="submit" NAME="cmdSubmit" VALUE="Submit">

How it looks:



JAVASCRIPT TEXT OBJECT

The JavaScript Text Object is a property of the form object. Example code:

<INPUT TYPE=TEXT NAME="firstname">

Here's how it looks:

JAVASCRIPT TEXTAREA OBJECT

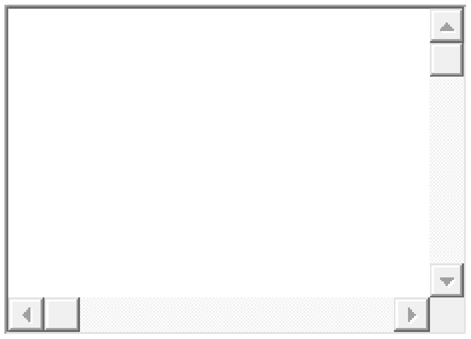
The JavaScript TextArea Object is a property of the form object. Textarea syntax is:

<TEXTAREA NAME="" ROWS="10" COLS="40" onBlur="blurHandlerRouting">

Displayed Text

</TEXTAREA>

Here's how it looks:



The NAME, ROWS, and COLS values are required. Three optional handlers may be specified which are "onBlur", "onChange", and "onSelect". If these handlers are used, the appropriate function must exist to support the action.

JAVA BUILT-IN OBJECT

Number Methods

The Number object contains only the default methods that are part of every object's definition.

Method	Description
constructor()	Returns the function that created this object's instance. By default this is the Number object.
toExponential()	Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
toFixed()	Formats a number with a specific number of digits to the right of the decimal.
toLocaleString()	Returns a string value version of the current number in a format that may vary according to a browser's locale settings.
toPrecision()	Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
toString()	Returns the string representation of the number's value.
valueOf()	Returns the number's value.

Boolean Methods

Here is a list of each method and its description.

Method	Description
toSource()	Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
toString()	Returns a string of either "true" or "false" depending upon the value of the object.
valueOf()	Returns the primitive value of the Boolean object.

String Methods

Here is a list of each method and its description.

Method	Description
charAt()	Returns the character at the specified index.
charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
concat()	Combines the text of two strings and returns a new string.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
length()	Returns the length of the string.
match()	Used to match a regular expression against a string.

replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
search()	Executes the search for a match between a regular expression and a specified string.
slice()	Extracts a section of a string and returns a new string.
split()	Splits a String object into an array of strings by separating the string into substrings.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring()	Returns the characters in a string between two indexes into the string.
toLocaleLowerCase()	The characters within a string are converted to lower case while respecting the current locale.
toLocaleUpperCase()	The characters within a string are converted to upper case while respecting the current locale.
toLowerCase()	Returns the calling string value converted to lower case.
toString()	Returns a string representing the specified object.
toUpperCase()	Returns the calling string value converted to uppercase.
valueOf()	Returns the primitive value of the specified object.

String HTML wrappers

Here is a list of each method which returns a copy of the string wrapped inside the appropriate HTML tag.

Method	Description

anchor()	Creates an HTML anchor that is used as a hypertext target.
big()	Creates a string to be displayed in a big font as if it were in a <big> tag.
blink()	Creates a string to blink as if it were in a <blink> tag.
bold()	Creates a string to be displayed as bold as if it were in a <b> tag.
fixed()	Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
fontcolor()	Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.
fontsize()	Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
italics()	Causes a string to be italic, as if it were in an <i> tag.
link()	Creates an HTML hypertext link that requests another URL.
small()	Causes a string to be displayed in a small font, as if it were in a <small> tag.
strike()	Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
sub()	Causes a string to be displayed as a subscript, as if it were in a <sub> tag
sup()	Causes a string to be displayed as a superscript, as if it were in a <sup> tag

Array Methods

Method	**Description**
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

Date Methods:

Here is a list of each method and its description.

Method	Description
Date()	Returns today's date and time
getDate()	Returns the day of the month for the specified date according to local time.
getDay()	Returns the day of the week for the specified date according to local time.
getFullYear()	Returns the year of the specified date according to local time.
getHours()	Returns the hour in the specified date according to local time.

getMilliseconds()	Returns the milliseconds in the specified date according to local time.
getMinutes()	Returns the minutes in the specified date according to local time.
getMonth()	Returns the month in the specified date according to local time.
getSeconds()	Returns the seconds in the specified date according to local time.
getTime()	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.
getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.
getUTCDay()	Returns the day of the week in the specified date according to universal time.
getUTCFullYear()	Returns the year in the specified date according to universal time.
getUTCHours()	Returns the hours in the specified date according to universal time.
getUTCMilliseconds()	Returns the milliseconds in the specified date according to universal time.
getUTCMinutes()	Returns the minutes in the specified date according to universal time.
getUTCMonth()	Returns the month in the specified date according to universal

	time.
getUTCSeconds()	Returns the seconds in the specified date according to universal time.
getYear()	**Deprecated -** Returns the year in the specified date according to local time. Use getFullYear instead.
setDate()	Sets the day of the month for a specified date according to local time.
setFullYear()	Sets the full year for a specified date according to local time.
setHours()	Sets the hours for a specified date according to local time.
setMilliseconds()	Sets the milliseconds for a specified date according to local time.
setMinutes()	Sets the minutes for a specified date according to local time.
setMonth()	Sets the month for a specified date according to local time.
setSeconds()	Sets the seconds for a specified date according to local time.
setTime()	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
setUTCDate()	Sets the day of the month for a specified date according to universal time.
setUTCFullYear()	Sets the full year for a specified date according to universal time.
setUTCHours()	Sets the hour for a specified date according to universal time.
setUTCMilliseconds()	Sets the milliseconds for a specified date according to

	universal time.
setUTCMinutes()	Sets the minutes for a specified date according to universal time.
setUTCMonth()	Sets the month for a specified date according to universal time.
setUTCSeconds()	Sets the seconds for a specified date according to universal time.
setYear()	**Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
toDateString()	Returns the "date" portion of the Date as a human-readable string.
toGMTString()	**Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
toLocaleDateString()	Returns the "date" portion of the Date as a string, using the current locale's conventions.
toLocaleFormat()	Converts a date to a string, using a format string.
toLocaleString()	Converts a date to a string, using the current locale's conventions.
toLocaleTimeString()	Returns the "time" portion of the Date as a string, using the current locale's conventions.
toSource()	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
toString()	Returns a string representing the specified Date object.
toTimeString()	Returns the "time" portion of the Date as a human-readable

	string.
toUTCString()	Converts a date to a string, using the universal time convention.
valueOf()	Returns the primitive value of a Date object.

Date Static Methods:

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date( ) constructor itself:

Method	Description
Date.parse( )	Parses a string representation of a date and time and returns the internal millisecond representation of that date.
Date.UTC( )	Returns the millisecond representation of the specified UTC date and time.

Math Methods

Here is a list of each method and its description.

Method	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
asin()	Returns the arcsine (in radians) of a number.
atan()	Returns the arctangent (in radians) of a number.
atan2()	Returns the arctangent of the quotient of its arguments.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns the cosine of a number.

exp()	Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
floor()	Returns the largest integer less than or equal to a number.
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is, base exponent.
random()	Returns a pseudo-random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
sin()	Returns the sine of a number.
sqrt()	Returns the square root of a number.
tan()	Returns the tangent of a number.
toSource()	Returns the string "Math".

RegExp Methods:

Here is a list of each method and its description.

Method	**Description**
exec()	Executes a search for a match in its string parameter.
test()	Tests for a match in its string parameter.
toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
toString()	Returns a string representing the specified object.

USER DEFINED OBJECT IN JAVASCRIPT

An object is a real world entity that contains properties and behaviour. Properties are implemented as identifiers and behaviour is implemented using a set of methods. An object in JavaScript doesn’t contain any predefined type. In JavaScript the *new* operator is used to create a blank object with no properties. A constructor is used to create and initialize properties in JavaScript.

Note: In Java *new* operator is used to create the object and its properties and a constructor is used to initialize the properties of the created object.

An object can be created as shown below:

var obj = new Object( );

The properties of an object can be accessed using the dot (.) operator.

UNIT III

XML: Comparison with HTML – DTD – XML elements – Content creation – Attributes – Entities – XSL – XLINK – XPATH – XPOINTER – Namespaces – Applications – integrating XML with other applications.

What Is XML?

XML stands for Extensible Markup Language (often written as eXtensibleMarkup Language to justify the acronym). XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax used to define other domain-specific, semantic, structured markup languages.

DIFFERENCE BETWEEN XML AND HTML

- XML is the acronym from Extensible Markup Language (meta-language of noting/marking). XML is a resembling language with HTML. It was developed for describing data.
- The XML tags are not pre-defined in XML. You will have to create tags according to your needs.
- XML is self descriptive.
- XML uses DDT principle (Defining the Document Type) to formally describe the data.
- The main difference between XML and HTML: XML is not a substitute for HTML.

XML and HTML were developed with different purposes:

- XML was developed to describe data and to focalize on what the data represent.
- HTML was developed to display data about to focalize on the way that data looks.
- HTML is about displaying data, XML is about describing information.
- XML is extensible.

How can you use XML?

- XML can store data separately from HTML.
- XML can be used to store data inside the HTML documents.
- XML can be used as a format for exchanging information.
- XML can be used to store data in files and databases.

For example, in HTML a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML might look something like this:

```
<dt>Hot Cop
<dd> by Jacques Morali, Henri Belolo, and Victor Willis
<ul>
<li>Producer: Jacques Morali
<li>Publisher: PolyGram Records
<li>Length: 6:20
<li>Written: 1978
<li>Artist: Village People
</ul>
```

In XML the same data might be marked up like this:

```
<SONG>
<TITLE>Hot Cop</TITLE>
<COMPOSER>Jacques Morali</COMPOSER>
<COMPOSER>Henri Belolo</COMPOSER>
<COMPOSER>Victor Willis</COMPOSER>
<PRODUCER>Jacques Morali</PRODUCER>
<PUBLISHER>PolyGram Records</PUBLISHER>
<LENGTH>6:20</LENGTH>
<YEAR>1978</YEAR>
<ARTIST>Village People</ARTIST>
</SONG>
```

Instead of generic tags like <dt> and <li>, this listing uses meaningful tags like <SONG>, <TITLE>, <COMPOSER>, and <YEAR>. This has a number of advantages, including that it’s easier for a human to read the source code to determine what the author intended.

Cascading Style Sheets

Since XML allows arbitrary tags to be included in a document, there isn’t any way for the browser to know in advance how each element should be displayed. When you send a document to a user you also need to send along a style sheet that tells the browser how to format individual elements. One kind of style sheet you can use is a Cascading Style Sheet (CSS).

CSS, initially designed for HTML, defines formatting properties like font size, font family, font weight, paragraph indentation, paragraph alignment, and other styles that can be applied to particular elements.

XSL

XSL, the Extensible Style Language, is itself an XML application. XSL has two major parts. The first part defines a vocabulary for transforming XML documents. This part of XSL includes XML tags for trees, nodes, patterns, templates, and other elements needed for matching and transforming XML documents from one markup vocabulary to another (or even to the same one in a different order).

The second part of XSL defines an XML vocabulary for formatting the transformed XML document produced by the first part. This includes XML tags for formatting objects including pagination, blocks, characters, lists, graphics, boxes, fonts, and more. A typical XSL style sheet is shown in Listing 2-12:

An XSL style sheet

```
<?xml version=”1.0”?>
<xsl:stylesheet
xmlns:xsl=”http://www.w3.org/TR/WD-xsl”
xmlns:fo=”http://www.w3.org/TR/WD-xsl/FO”
result-ns=”fo”>
<xsl:template match=”/”>
```

```
<fo:basic-page-sequence >
<xsl:apply-templates/>
</fo:basic-page-sequence>
</xsl:template>
<xsl:template match=”ATOM”>
<fo:block font-size=”10pt” font-family=”serif”
space-before=”12pt”>
<xsl:value-of select=”NAME”/>
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

XLL

The Extensible Linking Language, XLL, defines a new, more general kind of link called an XLink. XLinks accomplish everything possible with HTML’s URL-based hyperlinks and anchors. However, any element can become a link, not just A elements. For instance a footnote element can link directly to the text of the note like this:

```
<footnote xlink:form=”simple” href=”footnote7.xml”>7</footnote>
```

Furthermore, XLinks can do a lot of things HTML links can’t. XLinks can be bidirectional so readers can return to the page they came from. XLinks can link toarbitrary positions in a document. XLinks can embed text or graphic data inside a document rather than requiring the user to activate the link (much like HTML’s <IMG> tag but more flexible). In short, XLinks make hypertext even more powerful.

XSL - More Than a Style Sheet Language

XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT - Transformation

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

or:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

XSLT <xsl:template> Element

An XSL style sheet consists of one or more set of rules that are called templates.

A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

Ok, let's look at a simplified version of the XSL file from the previous chapter:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>MyCDCollection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="UTF-8"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the root of the XML source document.

The content inside the <xsl:template> element defines some HTML to write to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:value-of>** element to select values from the XML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
  <cd>
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <country>USA</country>
    <company>RCA</company>
    <price>9.90</price>
    <year>1982</year>
  </cd>
  <cd>
    <title>Still got the blues</title>
    <artist>Gary Moore</artist>
    <country>UK</country>
    <company>Virgin records</company>
    <price>10.20</price>
    <year>1990</year>
  </cd>
</catalog>
```

XSLT CODE

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <tr>
        <td><xsl:value-of select="catalog/cd/title"/></td>
        <td><xsl:value-of select="catalog/cd/artist"/></td>
      </tr>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XML - DTDs

Advertisements

Previous Page
Next Page

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD −

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
```

```
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code −

Start Declaration − Begin the XML declaration with the following statement.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
```

DTD − Immediately after the XML header, the *document type declaration*follows, commonly referred to as the DOCTYPE −

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name*to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]>**). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD −

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** is as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows −

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers, or FPIs**.

XML - Elements

Advertisements

Previous Page

Next Page

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

Syntax

Following is the syntax to write an XML element −

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where,

- **element-name** is the name of the element. The *name* its case in the start and end tags must match.

- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as −

```
name = "value"
```

name is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax −

```
<name attribute1 attribute2.../>
```

Following is an example of an XML document using various XML element −

```
<?xml version = "1.0"?>
```

```
<contact-info>
   <address category = "residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   <address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements −

- An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

XML - Attributes

Advertisements

Previous Page

Next Page

This chapter describes the **XML attributes**. Attributes are part of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

Syntax

An XML attribute has the following syntax −

```
<element-name attribute1 attribute2 >
....content..
< /element-name>
```

where *attribute1* and *attribute2* has the following form −

```
name = "value"
```

value has to be in double (" ") or single (' ') quotes. Here, *attribute1* and *attribute2* are unique attribute labels.

Attributes are used to add a unique label to an element, place the label in a category, add a Boolean flag, or otherwise associate it with some string of data. Following example demonstrates the use of attributes −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE garden [
   <!ELEMENT garden (plants)*>
   <!ELEMENT plants (#PCDATA)>
   <!ATTLIST plants category CDATA #REQUIRED>
]>

<garden>
   <plants category = "flowers" />
   <plants category = "shrubs">
   </plants>
</garden>
```

Attributes are used to distinguish among elements of the same name, when you do not want to create a new element for every situation. Hence, the use of an attribute can add a little more detail in differentiating two or more similar elements.

In the above example, we have categorized the plants by including attribute category and assigning different values to each of the elements. Hence, we have two categories of *plants*, one *flowers* and other *color*. Thus, we have two plant elements with different attributes.

You can also observe that we have declared this attribute at the beginning of XML.

Attribute Types

Following table lists the type of attributes −

Attribute Type	**Description**
StringType	It takes any literal string as a value. CDATA is a StringType. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.
TokenizedType	This is a more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized. The TokenizedType attributes are given as − • **ID** − It is used to specify the element as unique. • **IDREF** − It is used to reference an ID that has been named for another element. • **IDREFS** − It is used to reference all IDs of an element. • **ENTITY** − It indicates that the attribute will represent an external entity in the document. • **ENTITIES** − It indicates that the attribute will represent external entities in the document. • **NMTOKEN** − It is similar to CDATA with restrictions on what data can be part of the attribute. • **NMTOKENS** − It is similar to CDATA with restrictions on what data can be part of the attribute.

EnumeratedType	This has a list of predefined values in its declaration. out of which, it must assign one value. There are two types of enumerated attribute − • **NotationType** − It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document. • **Enumeration** − Enumeration allows you to define a specific list of values that the attribute value must match.

Element Attribute Rules

Following are the rules that need to be followed for attributes −

- An attribute name must not appear more than once in the same start-tag or empty-element tag.
- An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.
- Attribute values must not contain direct or indirect entity references to external entities.
- The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a less than sign (<)

XML - Namespaces

Advertisements

Previous Page

Next Page

A **Namespace** is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to a group. The Namespace is identified by URI(Uniform Resource Identifiers).

Namespace Declaration

A Namespace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below −

```
<element xmlns:name = "URL">
```

Syntax

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<cont:contact xmlns:cont = "www.tutorialspoint.com/profile">
   <cont:name>Tanmay Patil</cont:name>
   <cont:company>TutorialsPoint</cont:company>
   <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is **cont**, and the Namespace identifier (URI) as *www.tutorialspoint.com/profile*. This means, the element names and attribute names with the **cont** prefix (including the contact element), all belong to the *www.tutorialspoint.com/profile* namespace.

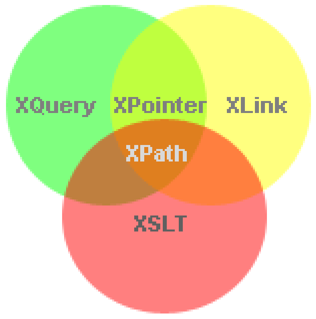
XML and XPath

❮ Previous Next ❯

What is XPath?

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.



- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery
- XPath is a W3C recommendation

XPath Path Expressions

XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath is Used in XSLT

XPath is a major element in the XSLT standard.

With XPath knowledge you will be able to take great advantage of XSL.

XPath Example

We will use the following XML document:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

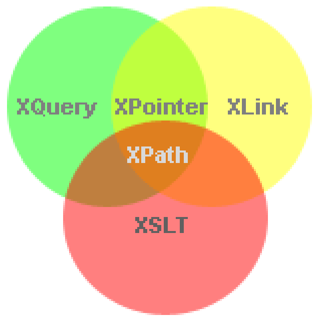
In the table below we have listed some XPath expressions and the result of the expressions:

XPath Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the books
/bookstore/book[last()]	Selects the last book element that is the child of the books
/bookstore/book[last()-1]	Selects the last but one book element that is the child of th
/bookstore/book[position()<3]	Selects the first two book elements that are children of the
//title[@lang]	Selects all the title elements that have an attribute named l
//title[@lang='en']	Selects all the title elements that have a "lang" attribute w
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element th with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the have a price element with a value greater than 35.00

XML, XLink and XPointer

❮ Previous Next ❯

XLink is used to create hyperlinks in XML documents.



- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files
- XLink is a W3C Recommendation

XLink Browser Support

There is no browser support for XLink in XML documents.

However, all major browsers support XLinks in SVG.

XLink Syntax

In HTML, the <a> element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.

Below is a simple example of how to use XLink to create links in an XML document:

```
<?xml version="1.0" encoding="UTF-8"?>

<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="https://www.w3schools.com">Visit
W3Schools</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit       W3C</homepage>
</homepages>
```

To get access to the XLink features we must declare the XLink namespace. The XLink namespace is: "http://www.w3.org/1999/xlink".

The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.

The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").

The xlink:href attribute specifies the URL to link to.

XLink Example

The following XML document contains XLink features:

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

<book title="HarryPotter">
  <description xlink:type="simple"xlink:href="/images/HPotter.gif"xlink:show="new">
      As his fifth year at Hogwarts School of Witchcraft and
        Wizardry approaches, 15-year-old Harry Potter is.......
  </description>
</book>

<book title="XQueryKickStart">
  <descriptionxlink:type="simple"xlink:href="/images/XQuery.gif"xlink:show="new">
        XQuery Kick Start delivers a concise introduction
                to the XQuery standard.......
  </description>
</book>

</bookstore>
```

Example explained:

- The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
- The xlink:type="simple" creates a simple "HTML-like" link
- The xlink:href attribute specifies the URL to link to (in this case - an image)
- The xlink:show="new" specifies that the link should open in a new window

XLink - Going Further

In the example above we have demonstrated simple XLinks. XLink is getting more interesting when accessing remote locations as resources, instead of standalone pages.

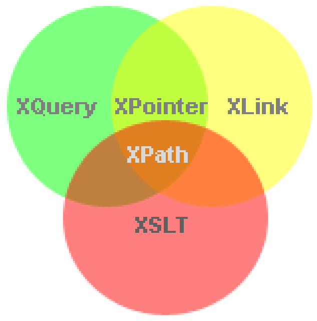
If we set the value of the xlink:show attribute to "embed", the linked resource should be processed inline within the page. When you consider that this could be another XML document you could, for example, build a hierarchy of XML documents.

You can also specify WHEN the resource should appear, with the xlink:actuate attribute.

XLink Attribute Reference

Attribute	**Value**	**Description**
xlink:actuate	onLoad onRequest other none	Defines when the linked resource is read and shown: • onLoad - the resource should be loaded and shown when the document l • onRequest - the resource is not read or shown before the link is clicked
xlink:href	*URL*	Specifies the URL to link to
xlink:show	embed new replace other none	Specifies where to open the link. Default is "replace"
xlink:type	simple extended locator arc resource title none	Specifies the type of link

XPointer



- XPointer allows links to point to specific parts of an XML document
- XPointer uses XPath expressions to navigate in the XML document
- XPointer is a W3C Recommendation

XPointer Browser Support

There is no browser support for XPointer. But XPointer is used in other XML languages.

XPointer Example

In this example, we will use XPointer in conjunction with XLink to point to a specific part of another document.

We will start by looking at the target XML document (the document we are linking to):

```
<?xml version="1.0" encoding="UTF-8"?>

<dogbreeds>

<dog breed="Rottweiler" id="Rottweiler">
  <picture url="https://dog.com/rottweiler.gif" />
  <history>The Rottweiler's ancestors were probably Roman drover dogs.....</history>
  <temperament>Confident, bold, alert and imposing, the Rottweiler is a popular choice for its ability to protect....</temperament>
</dog>

<dog breed="FCRetriever" id="FCRetriever">
  <picture url="https://dog.com/fcretriever.gif" />
  <history>One of the earliest uses of retrieving dogs was to help fishermen retrieve fish from the water....</history>
  <temperament>The flat-coated retriever is a sweet, exuberant,
     lively dog that loves to play and retrieve....</temperament>
</dog>
</dogbreeds>
```

Note that the XML document above uses id attributes on each element!

So, instead of linking to the entire document (as with XLink), XPointer allows you to link to specific parts of the document. To link to a specific part of a page, add a number sign (#) and an XPointer expression after the URL in the xlink:href attribute, like this: xlink:href="https://dog.com/dogbreeds.xml#xpointer(id('Rottweiler'))". The expression refers to the element in the target document, with the id value of "Rottweiler".

XPointer also allows a shorthand method for linking to an element with an id. You can use the value of the id directly, like this: xlink:href="https://dog.com/dogbreeds.xml#Rottweiler".

The following XML document contains links to more information of the dog breed for each of my dogs:

```
<?xml version="1.0" encoding="UTF-8"?>

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog>
  <description>
     Anton is my favorite dog. He has won a lot of.....
  </description>
  <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#Rottweiler">
     Fact about Rottweiler
  </fact>
</mydog>

<mydog>
  <description>
     Pluto is the sweetest dog on earth......
  </description>
  <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#FCRetriever">
     Fact about flat-coated Retriever
  </fact>
</mydog>

</mydogs>
```

Applications of XML

There are already hundreds of serious applications of XML.

XHTML

W3C's XMLization of HTML 4.0. Example XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head><title>Hello world!</title></head>
  <body><p>foobar</p></body>
</html>
```

CML

Chemical Markup Language. Example CML document snippet:

```
<molecule id="METHANOL">
  <atomArray>
    <stringArray builtin="elementType">C O H H H H</stringArray>
    <floatArray builtin="x3" units="pm">
      -0.748 0.558 -1.293 -1.263 -0.699 0.716
    </floatArray>
  </atomArray>
</molecule>
```

WML

Wireless Markup Language for WAP services:

```
<?xml version="1.0"?>
<wml>
   <card id="Card1" title="Wap-UK.com">
     <p>
       Hello World
     </p>
   </card>
</wml>
```

ThML

Theological Markup Language:

```
<h3 class="s05" id="One.2.p0.2">Having a Humble Opinion of Self</h3>
<p class="First" id="One.2.p0.3">EVERY man naturally desires knowledge
 <note place="foot" id="One.2.p0.4">
  <p class="Footnote" id="One.2.p0.5"><added id="One.2.p0.6">
    <name id="One.2.p0.7">Aristotle</name>, Metaphysics, i. 1.
  </added></p>
 </note>;
 but what good is knowledge without fear of God? Indeed a humble
 rustic who serves God is better than a proud intellectual who
 neglects his soul to study the course of the stars.
 <added id="One.2.p0.8"><note place="foot" id="One.2.p0.9">
   <p class="Footnote" id="One.2.p0.10">
    Augustine, Confessions V. 4.
   </p>
 </note></added>
</p>
```

UNIT IV

JSP Fundamentals: Basics – Directive basics – Page directive – The taglib directive – The include directive – JSP Standard Actions – Java Beans – Error Handling.

JSP Fundamentals :

- What is JavaServer Pages? JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with<% and ends with %>.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
- Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP, etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP:

Following is the list of other advantages of using JSP over other technologies:

- **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.

Setting up Java Development Kit

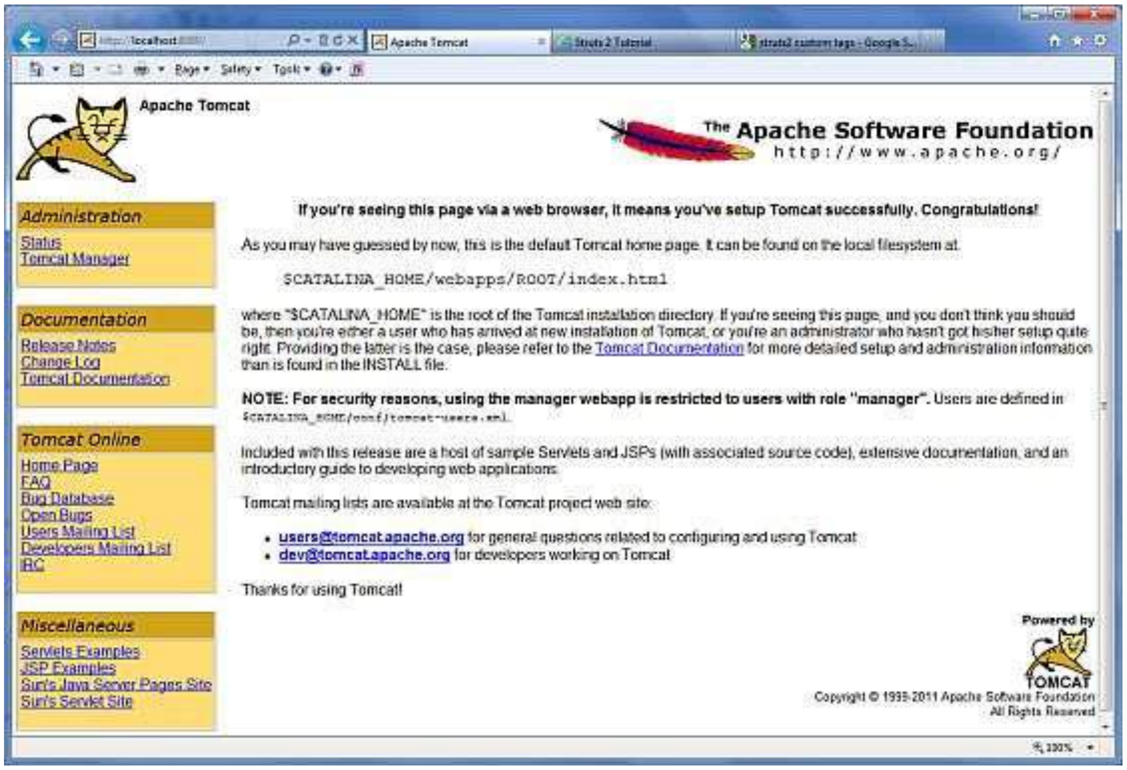
- ➢ This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.
- ➢ You can download SDK from Oracle's Java site: Java SE Downloads.
- ➢ Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
- ➢ If you are running Windows and installed the SDK in C:\jdk1.5.0_20, you would put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.5.0_20
```

- ➢ Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.
- ➢ Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

Setting up Web Server: Tomcat

- A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers are freely downloadable and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine:
- Download latest version of Tomcat from http://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-5.5.29 on windows, or /usr/local/apache-tomcat-5.5.29 on Linux/Unix and create CATALINA_HOME environment variable pointing to these locations.
- Tomcat can be started by executing the following commands on windows machine: C:\apache-tomcat-5.5.29\bin\startup.bat
- After a successful startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine then it should display following result:



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in $CATALINA_HOME/conf/tomcat-users.xml

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat
- dev@tomcat.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Copyright © 1999-2011 Apache Software Foundation
All Rights Reserved

Tomcat can be stopped by executing the following commands on windows machine:

```
C:\apache-tomcat-5.5.29\bin\shutdown
```

Setting up CLASSPATH

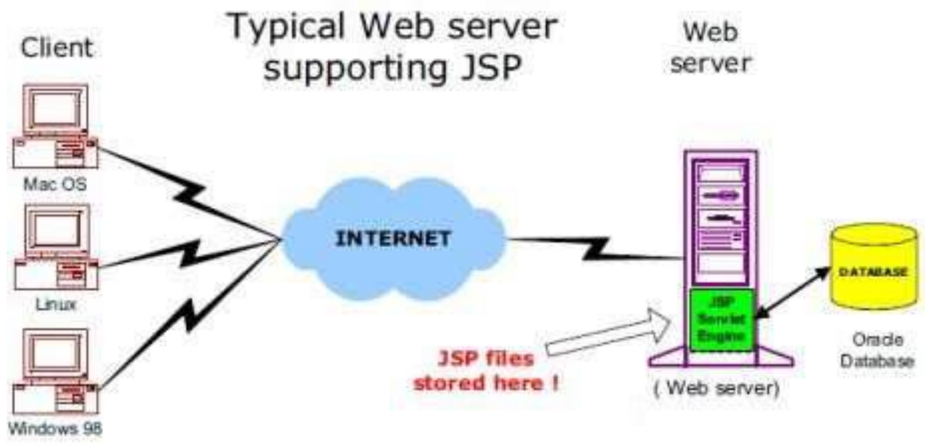
- Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.
- For Windows, you need to put the following lines in your C:\autoexec.bat file.

```
set CATALINA=C:\apache-tomcat-5.5.29
set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%
```

JSP ARCHITECTURE

The web server needs a JSP engine ie. container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs. Following diagram shows the position of JSP container and JSP files in a Web Application.



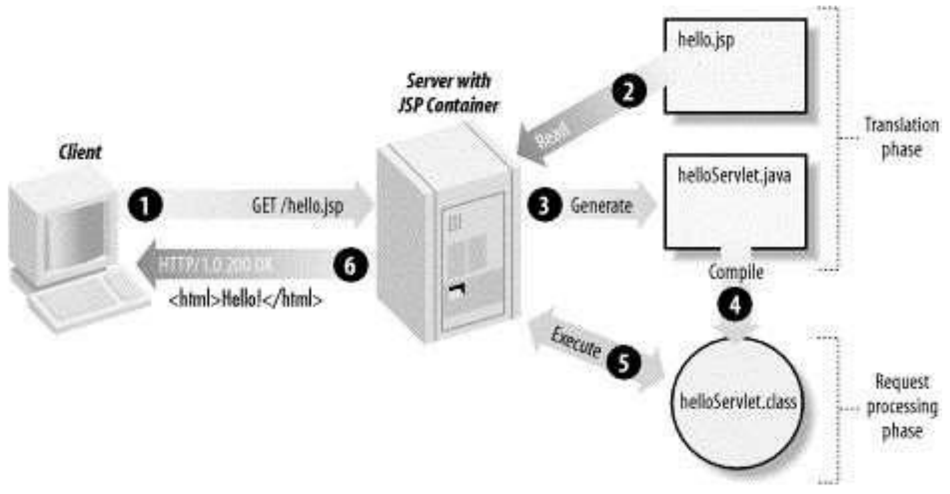
JSP Processing:

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements

and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.
- All the above mentioned steps can be shown below in the following diagram:



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

JSP LIFECYCLE

The following are the paths followed by a JSP

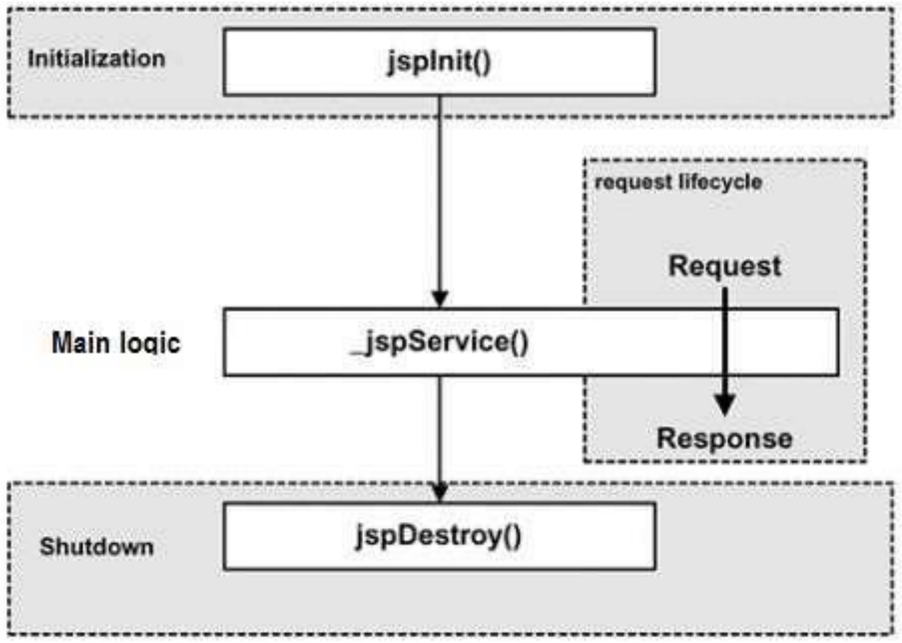
 Compilation

 Initialization

 Execution

 Cleanup

The four major phases of JSP life cycle are very similar to Servlet Life Cycle and they are as follows:



JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

 Parsing the JSP.

 Turning the JSP into a servlet.

 Compiling the servlet.

JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
// Initialization code...
}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

```
void _jspService(HttpServletRequest request,
HttpServletResponse response)
{
// Service handling code...
}
```

The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

```
public void jspDestroy()
{
// Your cleanup code goes here.
}
```

JSP SYNTAX

This chapter will give basic idea on simple syntax (ie. elements) involved with JSP development:

The Scriptlet:

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet:

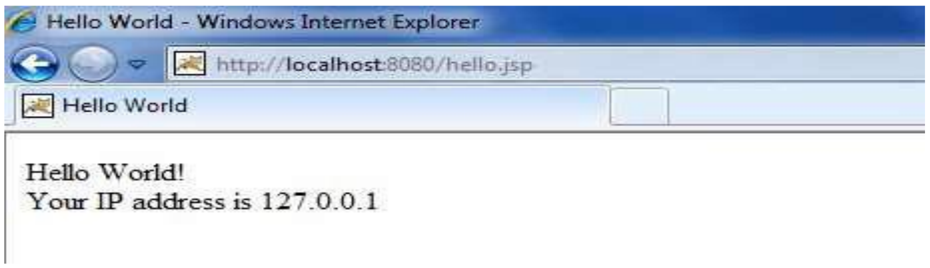
```
<% code fragment %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:scriptlet>
code fragment
</jsp:scriptlet>
```

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP:

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```


Hello World - Windows Internet Explorer
http://localhost:8080/hello.jsp
Hello World

```
Hello World!
Your IP address is 127.0.0.1
```

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

```
<%! declaration; [ declaration; ]+ ... %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:declaration>
    code fragment
</jsp:declaration>
```

Following is the simple example for JSP Declarations:

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP Expression:

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression:

```
<%= expression %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:expression>
expression
</jsp:expression>
```

Following is the simple example for JSP Expression:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

This would generate following result: Today's date: 11-Sep-2010 21:24:25

JSP Comments:

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

Following is the syntax of JSP comments:

```
<%-- This is JSP comment --%>
```

Following is the simple example for JSP Comments:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>
```

JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

```
<%@ directive attribute="value" %>
```

There are three types of directive tag:

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

JSP Actions:

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard:

```
<jsp:action_name attribute="value" />
```

Action elements are basically predefined functions and there are following JSP actions available:

Syntax	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.

Control-Flow Statements:

JSP provides full power of Java to be embedded in your web application. You can use all the APIs and building blocks of Java in your JSP programming including decision making statements, loops etc.

Decision-Making Statements:

The **if...else** block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
<p> Today is weekend</p>
<% } else { %>
<p> Today is not weekend</p>
<% } %>
</body>
</html>
```

Now look at the following **switch...case** block which has been written a bit differentlty using **out.println()**and inside Scriptletas:

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day) {
case 0:
out.println("It\'s Sunday.");
break;
case 1:
out.println("It\'s Monday.");
break;
case 2:
out.println("It\'s Tuesday.");
break;
case 3:
out.println("It\'s Wednesday.");
break;
case 4:
out.println("It\'s Thursday.");
break;
case 5:
out.println("It\'s Friday.");
break;
default:
out.println("It's Saturday.");
}
%>
</body>
</html>
```

This would produce following result: It's Wednesday.

Loop Statements:

You can also use three basic types of looping blocks in Java: **for, while, and do…while** blocks in your JSP programming.

Let us look at the following **for** loop example:

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
<font color="green" size="<%= fontSize %>">
JSP Tutorial
</font><br />
<%}%>
</body>
</html>
```

This would produce following result:

```
JSP Tutorial
JSP Tutorial
JSP Tutorial
```

WHILE LOOP

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while ( fontSize <= 3){ %>
<font color="green" size="<%= fontSize %>">
JSP Tutorial
</font><br />
<%fontSize++;%>
<%}%>
</body>
</html>
```

This would also produce following result:

JSP Tutorial

JSP Tutorial

JSP Tutorial

JSP Operators

JSP supports all the logical and arithmetic operators supported by Java. Following table lists out all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associatively**
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right

JSP - DIRECTIVES

In this chapter, we will discuss Directives in JSP. These directives provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form −

```
<%@ directive attribute = "value" %>
```

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag −

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP - The page Directive

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive −

```
<%@ page attribute = "value" %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:directive.page attribute = "value" />
```

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class, interface or all the members of a package. It is similar to import keyword in java class or interface.

Example of import attribute

```
<html>
<body>
<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
</body>
</html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response. The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
<html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>
</body>   </html>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

```
<%@ page extends="org.apache.jasper.runtime.HttpJspBase" %>
```

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
<html>
<body>
<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

```
<%@ page language="java" %>
```

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded. If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive. The value of isThreadSafe value is true. If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it. If you make the value of isThreadSafe attribute like:

```
<%@ page isThreadSafe="false" %>
```

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

```
//index.jsp
<html>
<body>
<%@ page errorPage="myerrorpage.jsp" %>
 <%= 100/0 %>
</body>
</html>
```

10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
//myerrorpage.jsp
<html>
<body>
<%@ page isErrorPage="true" %>
 Sorry an exception occured!<br/>
The exception is: <%= exception %>
</body>
</html>
```

The include Directive

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include*** directives anywhere in your JSP page.

The general usage form of this directive is as follows −

```
<%@ include file = "relative url" >
```

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows −

```
<jsp:directive.include file = "relative url" />
```

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>
<%@ include file="header.html" %>
```

```
Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

```
<%@ taglib uri="uri" prefix = "prefixOfTag" >
```

Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows −

```
<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />
```

Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>
<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
<mytag:currentDate/>
</body>
</html>
```

Standard Tag(Action Element)

JSP specification provides **Standard**(Action) tags for use within your JSP pages. These tags are used to remove or eliminate scriptlet code from your JSP page because scriplet code are technically not recommended nowadays. It's considered to be bad practice to put java code directly inside your JSP page.

Standard tags begin with the jsp: prefix. There are many JSP Standard Action tag which are used to perform some specific task.

The following are some JSP Standard Action Tags available:

Action Tag	**Description**
jsp:forward	forward the request to a new page Usage : <jsp:forward page="Relative URL" />
jsp:useBean	instantiates a JavaBean Usage : <jsp:useBean id="beanId" />
jsp:getProperty	retrieves a property from a JavaBean instance. Usage : <jsp:useBean id="beanId" ... /> ... <jsp:getProperty name="beanId" property="someProperty" .../> Where, **beanName** is the name of pre-defined bean whose property we want to access.
jsp:setProperty	store data in property of any JavaBeans instance. Usage : <jsp:useBean id="beanId" ... /> ... <jsp:setProperty name="beanId" property="someProperty" value="some value"/> Where, **beanName** is the name of pre-defined bean whose property we want to access.
jsp:include	includes the runtime response of a JSP page into the current page.
jsp:plugin	Generates client browser-specific construct that makes an OBJECT or EMBED tag for the Java Applets
jsp:fallback	Supplies alternate text if java plugin is unavailable on the client. You can print a message using this, if the included jsp plugin is not loaded.

jsp:element	Defines XML elements dynamically
jsp:attribute	defines dynamically defined XML element's attribute
jsp:body	Used within standard or custom tags to supply the tag body.
jsp:param	Adds parameters to the request object.
jsp:text	Used to write template text in JSP pages and documents. Usage : <jsp:text>Template data</jsp:text>

JSP - JAVABEANS

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes −

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class −

S.No.	Method & Description
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

JavaBeans Example

Consider a student class with few properties −

```
package com.tutorialspoint;
public class StudentsBean implements java.io.Serializable {
   private String firstName = null;
   private String lastName = null;
   private int age = 0;
   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

Accessing JavaBeans

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a **page, request, session** or **application based** on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Following example shows how to use the useBean action −

```
<html>
   <head>
      <title>useBean Example</title>
   </head>
      <body>
      <jsp:useBean id = "date" class = "java.util.Date" />
      <p>The date/time is <%= date %>
   </body>
</html>
```

You will receive the following result − −

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

Accessing JavaBeans Properties

Along with **<jsp:useBean...>** action, you can use the **<jsp:getProperty/>**action to access the get methods and the **<jsp:setProperty/>** action to access the set methods. Here is the full syntax −

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
   <jsp:setProperty name = "bean's id" property = "property name"
      value = "value"/>
   <jsp:getProperty name = "bean's id" property = "property name"/>
   ...........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the **get**or the **set** methods that should be invoked.

Following example shows how to access the data using the above syntax −

```
<html>
  <head>
    <title>get and set properties Example</title>
  </head>
    <body>
    <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
      <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>
      <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>
      <jsp:setProperty name = "students" property = "age" value = "10"/>
    </jsp:useBean>

    <p>Student First Name:
      <jsp:getProperty name = "students" property = "firstName"/>
    </p>
     <p>Student Last Name:
      <jsp:getProperty name = "students" property = "lastName"/>
    </p>
     <p>Student Age:
      <jsp:getProperty name = "students" property = "age"/>
    </p>
  </body>
</html>
```

Let us make the **StudentsBean.class** available in CLASSPATH. Access the above JSP. the following result will be displayed −

```
Student First Name: Zara
Student Last Name: Ali
Student Age: 10
```

JSP - EXCEPTION HANDLING

In this chapter. we will discuss how to handle exceptions in JSP. When you are writing a JSP code, you might make coding errors which can occur at any part of the code. There may occur the following type of errors in your JSP code −

Checked exceptions

A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.

Runtime exceptions

A runtime exception is an exception that probably could have been avoided by the programmer. As opposed to the checked exceptions, runtime exceptions are ignored at the time of compliation.

Errors

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

We will further discuss ways to handle run time exception/error occuring in your JSP code.

Using Exception Object

The exception object is an instance of a subclass of Throwable (e.g., java.lang. NullPointerException) and is only available in error pages. Following table lists out the important methods available in the Throwable class.

S.No.	Methods & Description
1	**public String getMessage()** Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	**public Throwable getCause()** Returns the cause of the exception as represented by a Throwable object.

3	**public String toString()** Returns the name of the class concatenated with the result of **getMessage()**.
4	**public void printStackTrace()** Prints the result of **toString()** along with the stack trace to **System.err**, the error output stream.
5	**public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public Throwable fillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

JSP gives you an option to specify **Error Page** for each JSP. Whenever the page throws an exception, the JSP container automatically invokes the error page.

Following is an example to specifiy an error page for a **main.jsp**. To set up an error page, use the **<%@ page errorPage = "xxx" %>** directive.

```
<%@ page errorPage = "ShowError.jsp" %>
<html>
  <head>
    <title>Error Handling Example</title>
  </head>
    <body>
    <%
      // Throw an exception to invoke the error page
      int x = 1;
      if (x == 1) {
        throw new RuntimeException("Error condition!!!");
      }
    %>
  </body>
```

```
</html>
```

We will now write one Error Handling JSP ShowError.jsp, which is given below. Notice that the error-handling page includes the directive **<%@ page isErrorPage = "true" %>**. This directive causes the JSP compiler to generate the exception instance variable.

```
<%@ page isErrorPage = "true" %>
<html>
  <head>
    <title>Show Error Page</title>
  </head>
  <body>
    <h1>Opps...</h1>
    <p>Sorry, an error occurred.</p>
    <p>Here is the exception stack trace: </p>
    <pre><% exception.printStackTrace(response.getWriter()); %></pre>
  </body>
</html>
```

Access the **main.jsp**, you will receive an output somewhat like the following −

```
java.lang.RuntimeException: Error condition!!!
......
Opps...
Sorry, an error occurred.
Here is the exception stack trace:
```

Using JSTL Tags for Error Page

You can make use of JSTL tags to write an error page **ShowError.jsp**. This page has almost same logic as in the above example, with better structure and more information −

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<%@page isErrorPage = "true" %>

<html>
  <head>
```

```
      <title>Show Error Page</title>
   </head>
      <body>
      <h1>Opps...</h1>
      <table width = "100%" border = "1">
         <tr valign = "top">
            <td width = "40%"><b>Error:</b></td>
            <td>${pageContext.exception}</td>
         </tr>
         <tr valign = "top">
            <td><b>URI:</b></td>
            <td>${pageContext.errorData.requestURI}</td>
         </tr>
         <tr valign = "top">
            <td><b>Status code:</b></td>
            <td>${pageContext.errorData.statusCode}</td>
         </tr>
         <tr valign = "top">
            <td><b>Stack trace:</b></td>
            <td>
               <c:forEach var = "trace"
                  items = "${pageContext.exception.stackTrace}">
                  <p>${trace}</p>
               </c:forEach>
            </td>
         </tr>
      </table>
   </body>
</html>
```

Access the main.jsp, the following will be generated −

Opps...

Error:	java.lang.RuntimeException: Error condition!!!
URI:	/main.jsp
Status code:	500

Stack trace:	org.apache.jsp.main_jsp._jspService(main_jsp.java:65)
	org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:68)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
	org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:722)

Using Try...Catch Block

If you want to handle errors within the same page and want to take some action instead of firing an error page, you can make use of the **try....catch**block.

Following is a simple example which shows how to use the try...catch block. Let us put the following code in main.jsp −

```
<html>
  <head>
    <title>Try...Catch Example</title>
  </head>
    <body>
    <%
      try {
        int i = 1;
        i = i / 0;
        out.println("The answer is " + i);
      }
      catch (Exception e) {
        out.println("An exception occurred: " + e.getMessage());
      }
    %>
  </body>
</html>
```

Access the main.jsp, it should generate an output somewhat like the following −

```
An exception occurred: / by zero
```

UNIT V

ASP: Introduction to ASP – Objects – Components – Working with HTML forms – Connecting to Microsoft SQL Server & MS–Access Database – SQL statements with connection object – Working with record sets.

INTRODUCTION TO ASP

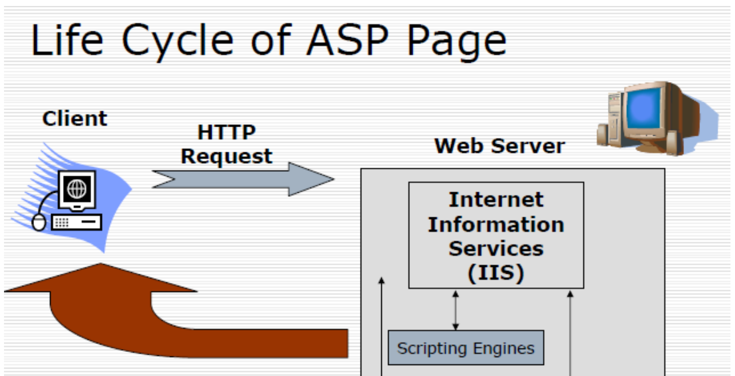
What is Active Server Pages (ASP)?

- As the name suggests, ASP represents pages that are executed on server side.
- ASP stands for Active Server Pages
- ASP is a program that runs inside IIS
- IIS stands for Internet Information Services
- IIS comes as a free component with Windows 2000 ,Windows XP and Windows 2000/2003 server.
- PWS (Personal Web Server) is a smaller - but fully functional - version of IIS
- PWS can be found on your Windows 95/98 CD
- When a client machine requests an ASP page, request is being sent to server. Server processes the request with the help of C:\WINDOWS\system32\inetsrv\asp.dll file and sends back the response to the client machine.

ASP Compatibility

- ASP is a Microsoft Technology
- To run IIS you must have Windows NT 4.0 or later
- To run PWS you must have Windows 95 or later
- Chili ASP is a technology that runs ASP without Windows OS
- Instant ASP is another technology that runs ASP without Windows

LIFE CYCLE OF ASP



What you can do with ASP?

 Dynamically edit, change or add any content of a Web page

 Respond to user queries or data submitted from HTML forms

 Access any data or databases and return the results to a browser

 Customize a Web page to make it more useful for individual users

 The advantages of using ASP instead of CGI and Perl, are those of simplicity and speed

 Provide security since your ASP code can not be viewed from the browser

 Clever ASP programming can minimize the network traffic.

The Basic Syntax Rule

An ASP file normally contains HTML tags, just like an HTML file. However, an ASP file can also contain **server scripts**, surrounded by the delimiters **<%** and **%>**. Server scripts are **executed on the server,** and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.

Write Output to a Browser

The response.write command is used to write output to a browser. The following example sends the text
"Hello World" to the browser:

```
<html>
<body>
<%response.write("Hello World!")%>
</body>
</html>
```

Variables

 A variable is used to store information.

 If the variable is declared outside a procedure it can be changed by any script in the ASP file. If the variable is declared inside a procedure, it is created and destroyed every time the procedure is executed.

Program With Variables

```
<html>
<body>
<%
dim h
h="Hello World"
response.write("Say: " & h)
%>
</body>
</html>
```

Variable Example

```
<%
Dim name, email, age
name=”John M”
email=”you@you.com”
age=35
response.write(“Your Name: “ & name & "<br>")
response.write(“Your Email: “ & email & "<br">)
response.Write(“Your age: “ & age)
%>
```

What is a Cookie?

 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.

 Each time the same computer access the page, the cookie will also be retrieved if the expiry date has future value.

How 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 10,2002#%>
```

How to Retrieve a Cookie Value?

 The "Request.Cookies" command is used to retrieve a cookie value.

 In the example below, we retrieve the value of the cookie named "firstname" and display it on a page:

```
<%fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)%>
```

Output: Firstname=Alex

A Cookie with Keys

 If a cookie contains a collection of multiple values, we say that the cookie has Keys.

 In the example below, we will create a cookie collection named "user". The "user" cookie has Keys that contains information about a user:

```
<%Response.Cookies("firstname")="Alex"
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
Response.Cookies("user")("age")="25" %>
```

OBJECTS IN ASP

An object is a collection of functionality and information. ASP has given us a handfull of Intrinsic Objects.

Intrinsic Objects

Object	**Used For**
Request	Getting information from the User
Response	Sending information to the User
Session	Storing information about a User's Session
Application	Sharing information for the entire application
Server	Accessing the server resources

Request Object

This object is mainly used to retrieve the information from the form in a HTML Page.

The Request Object has the following Collections:

- Form - To access value from a form submitted using POST method.
- QueryString - To access variables sent with URL after "?" or from a from submitted using GET method.
- Cookies - To access the value of a Cookie.
- ServerVariables - To access information from HTTP Headers.

The syntax to access the variables of any of these collections is Request.Collections("Variables"). But, you can also use Request("Variables") instead.

When we use the Form collection, we need to pass the name of the element that we create in HTML Form as the Variable to the Request Object.

For example:

If you have created a text box like one below:

<input type="text" name="Text1" value="">
then, your request statement should be,
Request.Form("Text1") or Request("Text1").

Likewise, when using Querystring you need to pass the variable name used in the URL or the name of the element specified in the form.(See the eg. below)

We can use the cookies collection to retrieve any cookies values set before. Again the variable should be the variable name used to set cookie value.

The last collection in the Request Object is ServerVariable. This is used to access the information from the HTTP headers like the remote user IP address, TCP/IP server port etc. We have a set of pre-defined variables to pass with the request object.

Example:

In the below form, enter values in the name and email text box and click the Submit button,to see how request object is working.

Name:
EMail :
Submit Clear

Response Object

This object is used to send information to the user (i.e. to the browser).

The most used methods of Response object are:

- Write - Used to send information to be displayed on the browser.
- Redirect - Used to send the user to a new Page.

The syntax to use these method is Response.Method

Eg: **Response.Redirect("newpage.html")**

The following statement will write the string inside paranthesis on the browser screen.

Statement

<% Response.Write("<font color=red>Text from Write Method</font>")%>

Output

Text from Write Method

Session Object

This object is used to store information with a scope to that user session. The information stored are maintained even when the user moves through various pages in the web application. The session object has two properties.

- SessionID - Created by the web application and sent as a cookie to client.
- TimeOut - To set Session timeout period.

The session object has one method, Abandon. This method is used to explicitly close the session and hence destroying the session object.

You can create new variables with session scope using the following syntax:

Session("Variablename")=Value

And the same can be referred using the following syntax:

Session("Variablename").

The data you entered in the form(name and email) of previous example is stored has session variables.

The values entered in the previous form are:(if you have not entered any values you will see "Please enter values for the name and email field in the form" message).

Please enter values for the name and email field in the form.

I have stored the values has <%Session("Name")%> and <%Session("EMail")%> in the previous form. And for retrieving i use the same syntax , but with a "=" symbol prefix.

Click the below button to close the session. This will clear the session variables set earlier.Here, I am using the Session.Abandon Method to destroy the session object.

Click Here!

Since the values are cleared, it will display "Please enter values for name and email field in the previous form" message.

Application Object

This object is used to share information among the users of the web application. The variable becomes alive, when the first request to the application comes in.

This object is typically used to create variables that need to maintain application level scope.

The Application object has the following methods:

- Lock- To Lock the variable
- Unlock - To unlock the variable

Since the application variables can be accessed by all the users of the application, the lock and unlock method is used to avoid more than one user accessing the same variable.

You can create new variables with Application scope using the following syntax:
Application("Variablename")=Value.

And the same can be referred using the following syntax:
Application("Variablename").

For every set of data you enter in the form(name and email) of previous example, we have stored a counter. This counter is stored as an Application variable. This is common to the entire application.

See the Total number of Records entered below:

Total number of Records entered so far:

Enter new values in the previous form and submit the same. Then see the value in above line being incremented.

I am using Application("NumValues") to store the value and increment it by one for every set of data you enter. Before incrementing I use Application.Lock method and after incrementing I use Application.UnLock method.

This is how an Application variable is useful.

Server Object

This object gives access to Server components, its methods and properties.

This object has the following methods:

- CreateObject - An important method used to create instance of Server Components
- HTMLEncode - To HTML encode a string passed
- URLEncode - To URL encode the string.

The CreateObject is a very important method and it is widely used in most ASP Pages.The syntax to use any of these methods is Server.Method.

For Eg:

Server.CreateObject("ADODB.Connection")

If you remember, ASP can use any of the Active Server Components registered on the server. And we have a set of Active Server Components with IIS. We use Createobject method to access this components.

There are components like Data Access Component, Browser Capabilities Component, Ad Rotator Component etc.

Each component exposes certain objects and methods. We can use these methods in our ASP Pages.For instance let us see how we can utilise the Browser Capabilities Component. We will use the Server.CreateObject method to access various methods and properties of Browser Capabilites component.

Browser Capabilites Component is used to find out the capabilities of the browser. We can verify the capabilites of the browser and deliver content based on its capabilities. For example, we can verify whether a browser supports frames or not and depending on the result, you can direct the user to a page with frames or to a page without frames. (Use response.redirect method to direct the users different pages.Hope you remember this method.)

The below table displays the capabilites of your browser:

Property	**Result**
Browser Type	Netscape
Version	4.00
Frames	True
Tables	True
Cookies	True
JavaScript	True
VBScript	False
ActiveX	unknown

Source

```
<%Setmyb=Server.CreateObject("MSWC.BrowserType")%>
<tableborder=2bordercolor="#000080">
<THEAD>
<tr>
<th>Property</th><th>Result</th>
</tr>
</THEAD>
<TBODY>
<tr>
<td>BrowserType</td><td><%=myb.Browser%></td>
</tr>
<tr>
<td>Version</td><td><%=myb.Version%></td>
</tr>
<tr>
<td>Frames</td><td><%=myb.Frames%></td>
</tr>
<tr>
<td>Tables</td><td><%=myb.Tables%></td>
</tr>
<tr>
<td>Cookies</td><td><%=myb.Cookies%></td>
</tr>
<tr>
<td>JavaScript</td><td><%=myb.Javascript%></td>
</tr>
<tr>
<td>VBScript</td><td><%=myb.VBScript%></td>
</tr>
<tr>
<td>ActiveX</td><td><%=myb.ActiveX%></td>
</tr>
</tbody>
</table>
```

The first line of the source creates an instance of the Browser Capabilites component. Then I am using the various properties of this component and displaying its value in a table. Similarly, we can use any of the components installed with the server or we can create our own active server component and use them in our ASP pages.

The opportunity to reuse any of the server components set the stage open for ASP developers to build their own component and use them, thus leaving only your imagination as a limit.

ASP COMPONENTS

Ad Rotator

The Ad Rotator streamlines the process of setting up a delivery system for your banner ads. In a separate file, you store information regarding the banner. The component then delivers a randomly selected banner every time the page is loaded.

adrot.txt

```
REDIRECT adredir.asp
width400
height 400
border 0
*
\image\1.jpg
http://www.aspsite.com
the active server pages site
1
\image\2.jpg
http://www.cityauction.com
cityauction
1
```

adredir.asp

```
<html>
<head><title>home page</title></head>
<body>
<center><h1>welcome to our website!</h1></center>
<hr>
<%
setmyad=server.CreateObject("MSWC.AdRotator")
%>
<center><%=myad.GetAdvertisement("adrot.txt")
%>
```

```
</center>
</body>
</html>
```

Browser Capabilities

The Browser component lets you determine what browser a user is using and what features are supported by that browser.

Syntax

```
<%
SetMyBrow=Server.CreateObject("MSWC.BrowserType")
%>
```

Collaboration Data Objects (CDO)

Tied in with the IIS SMTP server, CDO lets you send and receive email. With CDO, for example, you can process a form without relying upon a Perl script and CGI.

Syntax

```
<%
Setnl=Server.CreateObject("MSWC.NextLink")
%>
```

Example Program

```
<%
dimnlSetnl=Server.CreateObject("MSWC.NextLink")
if(nl.GetListIndex("links.txt")>1)then Response.Write("<ahref='"&
nl.GetPreviousURL("links.txt")) Response.Write("'>PreviousPage</a>")
endif
Response.Write("<ahref='"&nl.GetNextURL("links.txt"))
Response.Write("'>NextPage</a>")
%>
```

Content Linking

This is a handy object for creating a linear or sequential pathway through your site or a subsection of the site. You maintain a simple text file that lists the files in the proper sequence. Simple *next*and *previous* links then can be added to the page, and a table of contents can be easily generated.

Content Rotator

If you have a need for rotating content, this will be a favorite component. It is easy to use and allows you to add dynamic content to any page without using a database. In a separate text file, you store chunks of HTML code that you want alternately dropped into a space on the page. The Content Rotator will display one of the chunks each time the page is reloaded.

Syntax

```
<%
Setcr=Server.CreateObject("MSWC.ContentRotator")
%>
```

ASP Content Rotator Component's Methods

Method	**Description**	**Example**
ChooseContent	Gets and displays a content string	<% dim cr Set cr=Server.CreateObject("MSWC.ContentRotator") response.write(cr.ChooseContent("text/textads.txt")) %> Output:
GetAllContent	Retrieves and displays all of the content strings in the text file	<% dim cr Set cr=Server.CreateObject("MSWC.ContentRotator")

		response.write(cr.GetAllContent("text/textads.txt")) %> Output: This is a great day!! Visit W3Schools.com

Database Access

Using this component, you can hook into a database to write contents to the browser screen and to create or update existing database files.

Third-Party Components

There are numerous third-party components—both free and fee-based—available for ASP. If you're running your own server, you can install components at will. Registering a .dll is often the extent of the installation, so a component can be a real time-saver. Instead of spending hours re-creating the wheel, check to see if a component exists to handle the task at hand.

WORKING WITH HTML FORMS

The Request.QueryString and Request.Form commands are used to retrieve user input from forms.

User Input

The Request object can be used to retrieve user information from forms.

User input can be retrieved with the Request.QueryString or Request.Form command.

Request.QueryString

The Request.QueryString command is used to collect values in a form with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

Example HTML form

```
<formmethod="get"action="simpleform.asp">
FirstName:<inputtype="text"name="fname"><br>
```

```
LastName:<inputtype="text"name="lname"><br><br>
<inputtype="submit"value="Submit">
</form>
```

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

https://www.w3schools.com/simpleform.asp?fname=Bill&lname=Gates

Assume that "simpleform.asp" contains the following ASP script:

```
<body>
Welcome
<%
response.write(request.querystring("fname"))
response.write(""&request.querystring("lname"))
%>
</body>
```

The browser will display the following in the body of the document:

Welcome Bill Gates

Request.Form

The Request.Form command is used to collect values in a form with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Example HTML form

```
<formmethod="post"action="simpleform.asp">
FirstName:<inputtype="text"name="fname"><br>
LastName:<inputtype="text"name="lname"><br><br>
<inputtype="submit"value="Submit">
</form>
```

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

https://www.w3schools.com/simpleform.asp

Assume that "simpleform.asp" contains the following ASP script:

```
<body>
Welcome
<%
response.write(request.form("fname"))
response.write(""&request.form("lname"))
%>
</body>
```

The browser will display the following in the body of the document:

```
Welcome Bill Gates
```

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

Accessing Microsoft Access databases in ASP using ADO

Introduction

Windows DNA provides a means to provide your user interface, business logic and data sources as separate services working together in harmony over a distributed environment. The browser has become an extremely powerful, yet simple method of providing the user interface, since it handles the network considerations and allows you to create rich user interfaces through simple scripting, HTML and style sheets.

Your database considerations can be taken care of simply through the use of SQLServer or the Microsoft Jet Engine, and your business logic - the guts of your application that processes the data from the database and sends it to the browser - can be simple ASP pages (enhanced with ActiveX controls if the fancy takes you).

Once you have the basics of ASP, HTML and VBScript the business logic and user interface are taken care of quickly and simply - but how do you use ASP to access your database and hence complete your 3-tier application? Read on...

Simple database Access using ADO and ASP

For this example we'll use Access .mdb databases - but we could just as easily use SQLServer by changing a single line (and of course, configuring the databases correctly).

We'll be assuming your application is ASP based running on Microsoft's IIS Webserver.

We use ADO since it is portable, widespread, and very, very simple.

The Connection

To access a database we first need to open a connection to it, which involves creating an ADO Connection object. We then specify the connection string and call the Connection object's Open method.

To open an Access database our string would look like the following:

```
Dim ConnectionString
ConnectionString = "DRIVER={Microsoft Access Driver (*.mdb)};" &_
        "DBQ=C:\MyDatabases\database.mdb;DefaultDir=;UID=;PWD=;"
```

where the database we are concerned with is located at C:\MyDatabases\database.mdb, and has no username or password requirements. If we wanted to use a different database driver (such as SQLServer) then we simply provide a different connection string.

To create the ADO Connection object simply Dim a variable and get the server to do the work.

```
Dim Connection
Set Connection = Server.CreateObject("ADODB.Connection")
```

Then to open the database we (optionally) set some of the properties of the Connection and call Open

```
Connection.ConnectionTimeout = 30
Connection.CommandTimeout = 80
Connection.Open ConnectionString
```

Check for errors and if everything is OK then we are on our way.

The Records

Next we probably want to access some records in the database. This is achieved via the ADO RecordSet object. Using this objects Open method we can pass in any SQL string that our database driver supports and receive back a set of records (assuming your are SELECTing records, and not DELETEing).

```
' Create a RecordSet Object
Dim rs
set rs = Server.CreateObject("ADODB.RecordSet")
' Retrieve the records
rs.Open "SELECT * FROM MyTable", Connection, adOpenForwardOnly, adLockOptimistic
```

adOpenForwardOnly is defined as 0 and specifies that we only wish to traverse the records from first to last. adLockOptimistic is defined as 3 and allows records to be modified.

If there were no errors we now have access to all records in the table "MyTable" in our database.

The final step is doing something with this information. We'll simply list it.

Hide Copy Code

```
' This will list all Column headings in the table
```

```
Dim item
For each item in rs.Fields
        Response.Write item.Name & "<br>"
next
' This will list each field in each record
while not rs.EOF
        For each item in rs.Fields
                Response.Write item.Value & "<br>"
        next
        rs.MoveNext
wend
End Sub
```

If we know the field names of the records we can access them using rs("field1") where field1 is the name of a field in the table.

Always remember to close your recordsets and Connections and free any resources associated with them

```
rs.Close
set rs = nothing
Connection.Close
Set Connection = nothing
```

DATABASE USING ASP

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "mydsn.mdb"
%>
<%
set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select rollno, sname, mark1, mark2, mark3, total from stu", conn
```

```
%>
<body align=center>
<br>
<br>
<center>student details</center>
<table border="1" align="center">
<tr><th>rollno</th>
<th>sname</th>
<th>mark1</th>
<th>mark2</th>
<th>mark3</th>
<th>total</th>
</tr>
<%while not rs.eof%>
<tr>
<td><%=rs(0)%></td>
<td><%=rs(1)%></td>
<td><%=rs(2)%></td>
<td><%=rs(3)%></td>
<td><%=rs(4)%></td>
<td><%=rs(5)%>
</td></tr>
<% rs.movenext %>
<% wend %>
</table></body>
```

RECORDSET OBJECT

The ADO Recordset object is used to hold a set of records from a database table. A Recordset object consist of records and columns (fields).

In ADO, this object is the most important and the one used most often to manipulate data from a database.

ProgID

```
set objRecordset=Server.CreateObject("ADODB.recordset")
```

When you first open a Recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are True.

Recordset objects can support two types of updating:

- **Immediate updating** - all changes are written immediately to the database once you call the Update method.
- **Batch updating** - the provider will cache multiple changes and then send them to the database with the UpdateBatch method.

In ADO there are 4 different cursor types defined:

- **Dynamic cursor** - Allows you to see additions, changes, and deletions by other users.
- **Keyset cursor -** Like a dynamic cursor, except that you cannot see additions by other users, and it prevents access to records that other users have deleted. Data changes by other users will still be visible.
- **Static cursor** - Provides a static copy of a recordset for you to use to find data or generate reports. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.
- **Forward-only cursor** - Allows you to only scroll forward through the Recordset. Additions, changes, or deletions by other users will not be visible.

The cursor type can be set by the CursorType property or by the CursorType parameter in the Open method.

Note: Not all providers support all methods or properties of the Recordset object.

Properties

Property	**Description**
AbsolutePage	Sets or returns a value that specifies the page number in the Recordset object
AbsolutePosition	Sets or returns a value that specifies the ordinal position of the current record in the Recordset object
ActiveCommand	Returns the Command object associated with the Recordset
ActiveConnection	Sets or returns a definition for a connection if the connection is closed, or the current Connection object if the connection is open
BOF	Returns true if the current record position is before the first record, otherwise false
Bookmark	Sets or returns a bookmark. The bookmark saves the position of the current record
CacheSize	Sets or returns the number of records that can be cached
CursorLocation	Sets or returns the location of the cursor service
CursorType	Sets or returns the cursor type of a Recordset object
DataMember	Sets or returns the name of the data member that will be retrieved from the object referenced by the DataSource property
DataSource	Specifies an object containing data to be represented as a Recordset object
EditMode	Returns the editing status of the current record
EOF	Returns true if the current record position is after the last record, otherwise false
Filter	Sets or returns a filter for the data in a Recordset object
Index	Sets or returns the name of the current index for a Recordset object
LockType	Sets or returns a value that specifies the type of locking when editing a record in a Recordset
MarshalOptions	Sets or returns a value that specifies which records are to be returned to the server

MaxRecords	Sets or returns the maximum number of records to return to a Recordset object from a query
PageCount	Returns the number of pages with data in a Recordset object
PageSize	Sets or returns the maximum number of records allowed on a single page of a Recordset object
RecordCount	Returns the number of records in a Recordset object
Sort	Sets or returns the field names in the Recordset to sort on
Source	Sets a string value or a Command object reference, or returns a String value that indicates the data source of the Recordset object
State	Returns a value that describes if the Recordset object is open, closed, connecting, executing or retrieving data
Status	Returns the status of the current record with regard to batch updates or other bulk operations
StayInSync	Sets or returns whether the reference to the child records will change when the parent record position changes

Methods

Method	**Description**
AddNew	Creates a new record
Cancel	Cancels an execution
CancelBatch	Cancels a batch update
CancelUpdate	Cancels changes made to a record of a Recordset object
Clone	Creates a duplicate of an existing Recordset
Close	Closes a Recordset
CompareBookmarks	Compares two bookmarks
Delete	Deletes a record or a group of records
Find	Searches for a record in a Recordset that satisfies a specified criteria
GetRows	Copies multiple records from a Recordset object into a two-dimensional

	array
GetString	Returns a Recordset as a string
Move	Moves the record pointer in a Recordset object
MoveFirst	Moves the record pointer to the first record
MoveLast	Moves the record pointer to the last record
MoveNext	Moves the record pointer to the next record
MovePrevious	Moves the record pointer to the previous record
NextRecordset	Clears the current Recordset object and returns the next Recordset object by looping through a series of commands
Open	Opens a database element that gives you access to records in a table, the results of a query, or to a saved Recordset
Requery	Updates the data in a Recordset by re-executing the query that made the original Recordset
Resync	Refreshes the data in the current Recordset from the original database
Save	Saves a Recordset object to a file or a Stream object
Seek	Searches the index of a Recordset to find a record that matches the specified values
Supports	Returns a boolean value that defines whether or not a Recordset object supports a specific type of functionality
Update	Saves all changes made to a single record in a Recordset object
UpdateBatch	Saves all changes in a Recordset to the database. Used when working in batch update mode

Events

Event	**Description**
EndOfRecordset	Triggered when you try to move to a record after the last record
FetchComplete	Triggered after all records in an asynchronous operation have been fetched
FetchProgress	Triggered periodically in an asynchronous operation, to state how many

	more records that have been fetched
FieldChangeComplete	Triggered after the value of a Field object change
MoveComplete	Triggered after the current position in the Recordset has changed
RecordChangeComplete	Triggered after a record has changed
RecordsetChangeComplete	Triggered after the Recordset has changed
WillChangeField	Triggered before the value of a Field object change
WillChangeRecord	Triggered before a record change
WillChangeRecordset	Triggered before a Recordset change
WillMove	Triggered before the current position in the Recordset changes