MallaReddy Institute of Technology & Science

Maisammaguda (V),GundlaPochampally (M), Medchal District – 500 100 (T.S)

WEB TECHNOLOGIES

COURSE FILE

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
(2022-2023)

Faculty in charge **HOD-CSE**

Dr. V Senthil kumar **Dr. T Srikanth**

Assistant Professor

Mrs. K Mamatha
Assistant Professor

R18 B.Tech. CSE Syllabus JNTU HYDERABAD

UNIT- I

Introduction to PHP: Declaring variables, data types, arrays, strings, operators, expressions, control structures, functions, Reading data from web form controls like text boxes, radio buttons, lists etc., Handling File Uploads. Connecting to database (MySQL as reference), executing simple queries, handling results, Handling sessions and cookies

File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

UNIT- II

HTML Common tags- List, Tables, images, forms, Frames; Cascading Style sheets;

XML: Introduction to XML, Defining XML tags, their attributes and values, Document Type Definition, XML Schemes, Document Object Model, XHTML Parsing XML Data – DOM and SAX Parsers in java.

UNIT - III

Introduction to Servlets: Common Gateway Interface (CGt), Life cycle of a Servlet, deploying a servlet, The Servlet API, Reading Servlet parameters, Reading Initialization parameters, Handling Http Request & Responses, Using Cookies and Sessions, connecting to a database using JDBC.

UNIT - IV

Introduction to JSP: The Anatomy of a JSP Page, JSP Processing, Declarations, Directives, Expressions, Code Snippets, implicit objects, Using Beans in JSP Pages, Using Cookies and session for session tracking, connecting to database in JSP.

UNIT - V

Client-side Scripting: Introduction to Javascript, Javascript language – declaring variables, scope of variables, functions. event handlers (onclick, onsubmit etc.), Document Object Model, Form validation.

TEXT BOOKS:

1. Web Technologies, Uttam K Roy, Oxford University Press
2. The Complete Reference PHP — Steven Holzner, Tata McGraw-Hill

REFERENCE BOOKS

1. Web Programming, building internet applications, Chris Bates 2″ edition, Wiley Dreamtech
2. Java Server Pages —Hans Bergsten, SPD O'Reilly,Java Script, D.Flanagan
3. Beginning Web Programming-Jon Duckett WROX.
4. Programming world wide web, R.W.Sebesta, Fourth Edition, Pearson.
5. Internet and World Wide Web — How to program. Dietel and Nieto, Pearson.

UNIT I

Introduction to PHP

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, informix, and Microsoft SQL Server.

- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

PHP Variables

A variable in PHP is a name of memory location that holds data. A variable is a temporary storage that is used to store data temporarily.

In PHP, a variable is declared using $ sign followed by variable name.

Syntax of declaring a variable in PHP is given below:

$variablename=value;

PHP Variable: Declaring string, integer and float

Let's see the example to store string, integer and float values in PHP variables.

File: variable1.php

```
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

Output:

```
string is: hello string
integer is: 200
float is: 44.6
```

PHP Variable: Sum of two variables

File: variable2.php

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

Output:

```
11
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

File: variable3.php

```
<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)
```

```
echo "$a <br/> $_b";
?>
```

Output:

```
hello
hello
```

File: variableinvalid.php

```
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$4c <br/> $*d";
?>
```

Output:

```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable (T_VARIABLE)
 or '$' in C:\wamp\www\variableinvalid.php on line 2
```

PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types
2. Compound Types
3. Special Types

PHP Data Types: Scalar Types

There are 4 scalar data types in PHP.

1. boolean
2. integer
3. float
4. string

PHP Data Types: Compound Types

There are 2 compound data types in PHP.

1. array
2. object

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. resource
2. NULL

PHP Arrays

PHP array is an ordered map (contains value on the basis of key). It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

Less Code: We don't need to define multiple variables.

Easy to traverse: By the help of single loop, we can traverse all the elements of an array.
Sorting: We can sort the elements of array.

PHP Array Types

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

1st way:

```
$season=array("summer","winter","spring","autumn");
```

2nd way:

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

Example

File: array1.php

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

File: array2.php

```
<?php
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

PHP Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

1st way:

```
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
```

2nd way:

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
```

Example

File: arrayassociative1.php

```
<?php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary:".$salary["Kartik"]."<br/>";
?>
```

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

File: arrayassociative2.php

```
<?php
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

PHP String

A PHP string is a sequence of characters i.e. used to store and manipulate text. There are 4 ways to specify string in PHP.

- single quoted
- double quoted
- heredoc syntax
- newdoc syntax (since PHP 5.3)

Single Quoted PHP String

We can create a string in PHP by enclosing text in a single quote. It is the easiest way to specify string in PHP.

```
<?php
```

```
$str='Hello text within single quote';
echo $str;
?>
```

Output:

```
Hello text within single quote
```

We can store multiple line text, special characters and escape sequences in a single quoted PHP string.

```
<?php
$str1='Hello text
multiple line
text within single quoted string';
$str2='Using double "quote" directly inside single quoted string';
$str3='Using escape sequences \n in single quoted string';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
Hello text multiple line text within single quoted string
Using double "quote" directly inside single quoted string
Using escape sequences \n in single quoted string
```

Note: In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \' and backslash through \\ inside single quoted PHP strings.

```
<?php
$num1=10;
$str1='trying variable $num1';
$str2='trying backslash n and backslash t inside single quoted string \n \t';
$str3='Using single quote \'my quote\' and \\backslash';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
trying variable $num1
trying backslash n and backslash t inside single quoted string \n \t
Using single quote 'my quote' and \backslash
```

Double Quoted PHP String

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

```
<?php
$str="Hello text within double quote";
echo $str;
?>
```

Output:

```
Hello text within double quote
```

Now, you **can't use double quote directly** inside double quoted string.

```
<?php
$str1="Using double "quote" directly inside double quoted string";
echo $str1;
?>
```

Output:

```
Parse error: syntax error, unexpected 'quote' (T_STRING) in C:\wamp\www\string1.php on line
2
```

We **can store multiple line text, special characters and escape sequences** in a double quoted PHP string.

```
<?php
$str1="Hello text
multiple line
text within double quoted string";
$str2="Using double \"quote\" with backslash inside double quoted string";
$str3="Using escape sequences \n in double quoted string";
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
Hello text multiple line text within double quoted string
Using double "quote" with backslash inside double quoted string
Using escape sequences in double quoted string
```

In double quoted strings, **variable will be interpreted**.

```
<?php
$num1=10;
echo "Number is: $num1";
?>
```

Output:

```
Number is: 10
```

PHP String Functions

PHP provides various string functions to access and manipulate strings. A list of important PHP string functions are given below.

1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax

```
string strtolower ( string $string )
```

Example

```
<?php
$str="My name is KHAN";
```

```
$str=strtolower($str);
echo $str;
?>
```

Output:

```
my name is khan
```

2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

Syntax

```
string strtoupper ( string $string )
```

Example

```
<?php
$str="My name is KHAN";
$str=strtoupper($str);
echo $str;
?>
```

Output:

```
MY NAME IS KHAN
```

3) PHP ucfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

Syntax

```
string ucfirst ( string $str )
```

Example

```
<?php
$str="my name is KHAN";
$str=ucfirst($str);
echo $str;
?>
```

Output:

```
My name is KHAN
```

4) PHP lcfirst() function

The lcfirst() function returns string converting first character into lowercase. It doesn't change the case of other characters.

Syntax

```
1. string lcfirst ( string $str )
```

Example

```
<?php
$str="MY name IS KHAN";
$str=lcfirst($str);
echo $str;
```

```
?>
```

Output:

```
mY name IS KHAN
```

5) PHP ucwords() function

The ucwords() function returns string converting first character of each word into uppercase.

Syntax

```
string ucwords ( string $str )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=ucwords($str);
echo $str;
?>
```

Output:

```
My Name Is Sonoo Jaiswal
```

6) PHP strrev() function

The strrev() function returns reversed string.

Syntax

```
string strrev ( string $string )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=strrev($str);
echo $str;
?>
```

Output:

```
lawsiaj oonoS si eman ym
```

7) PHP strlen() function

The strlen() function returns length of the string.

Syntax

```
int strlen ( string $string )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=strlen($str);
echo $str;
?>
```

Output:

```
24
```

PHP Operators

PHP Operator is a symbol i.e used to perform operations on operands. For example:

1. $num=10+20;//+ is the operator and 10,20 are operands

In the above example, + is the binary + operator, 10 and 20 are operands and $num is variable.

PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Logical Operators
- String Operators
- Incrementing/Decrementing Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators
- Assignment Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- Unary Operators: works on single operands such as ++, -- etc.
- Binary Operators: works on two operands such as binary +, -, *, / etc.
- Ternary Operators: works on three operands such as "?:".

PHP Operators Precedence

Let's see the precedence of PHP operators with associativity.

Operators	**Additional Information**	**Associativity**
clone new	clone and new	non-associative
[	array()	left
**	arithmetic	right
++ -- ~ (int) (float) (string) (array) (object) (bool) @	increment/decrement and types	right
instanceof	types	non-associative
!	logical (negation)	right
* / %	arithmetic	left

+ - .	arithmetic and string concatenation	left
<<>>	bitwise (shift)	left
<<= >>=	comparison	non-associative
== != === !== <>	comparison	non-associative
&	bitwise AND	left
^	bitwise XOR	left
|	bitwise OR	left
&&	logical AND	left
||	logical OR	left
?:	ternary	left
= += -= *= **= /= .= %= &= |= ^= <<= >>= =>	assignment	right
And	logical	left
Xor	logical	left
Or	logical	left
,	many uses (comma)	left

PHP Expressions:
Expression is the combination of operators, variables and constants.
Control Statements
PHP If Else

PHP if else statement is used to test condition. There are various ways to use if statement in PHP.

- if
- if-else

- if-else-if
- nested if

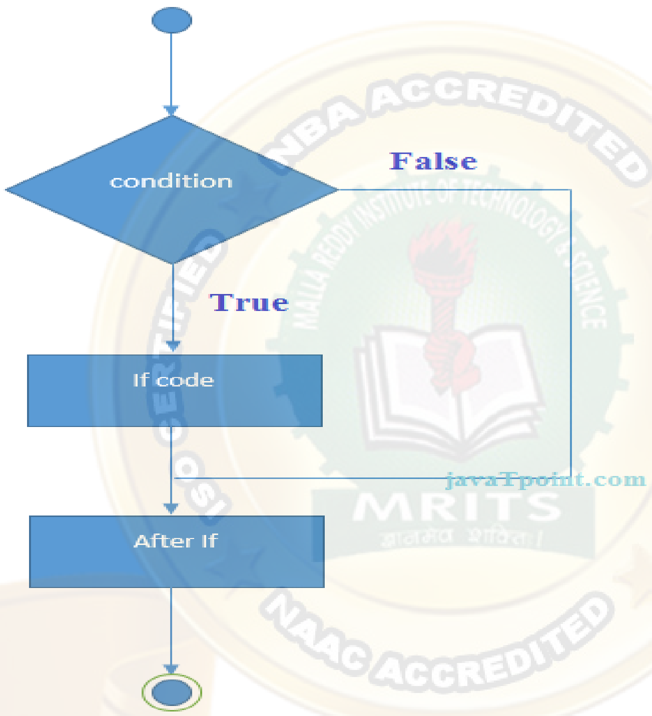
PHP If Statement

PHP if statement is executed if condition is true.

Syntax

```
if(condition){
//code to be executed
}
```

Flowchart



Example

```
<?php
$num=12;
if($num<100){
echo "$num is less than 100";
}
?>
```

Output:

```
12 is less than 100
```

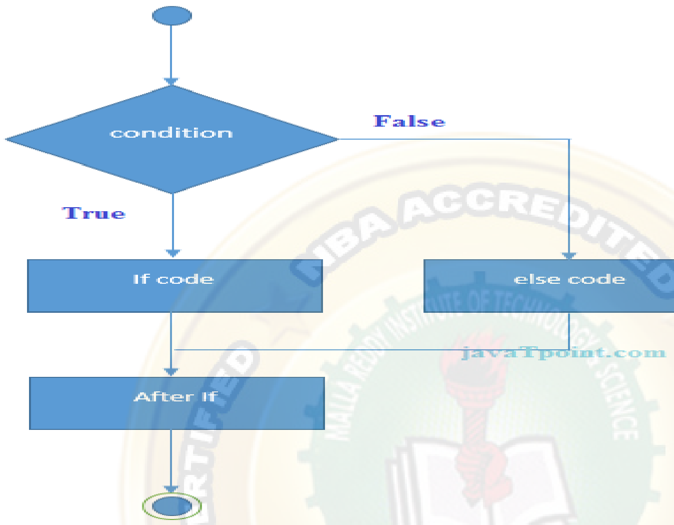
PHP If-else Statement

PHP if-else statement is executed whether condition is true or false.

Syntax

```
if(condition){
//code to be executed if true
}else{
//code to be executed if false
}
```

Flowchart



Example

```
<?php
$num=12;
if($num%2==0){
echo "$num is even number";
}else{
echo "$num is odd number";
}
?>
```

Output:

```
12 is even number
```

PHP Switch

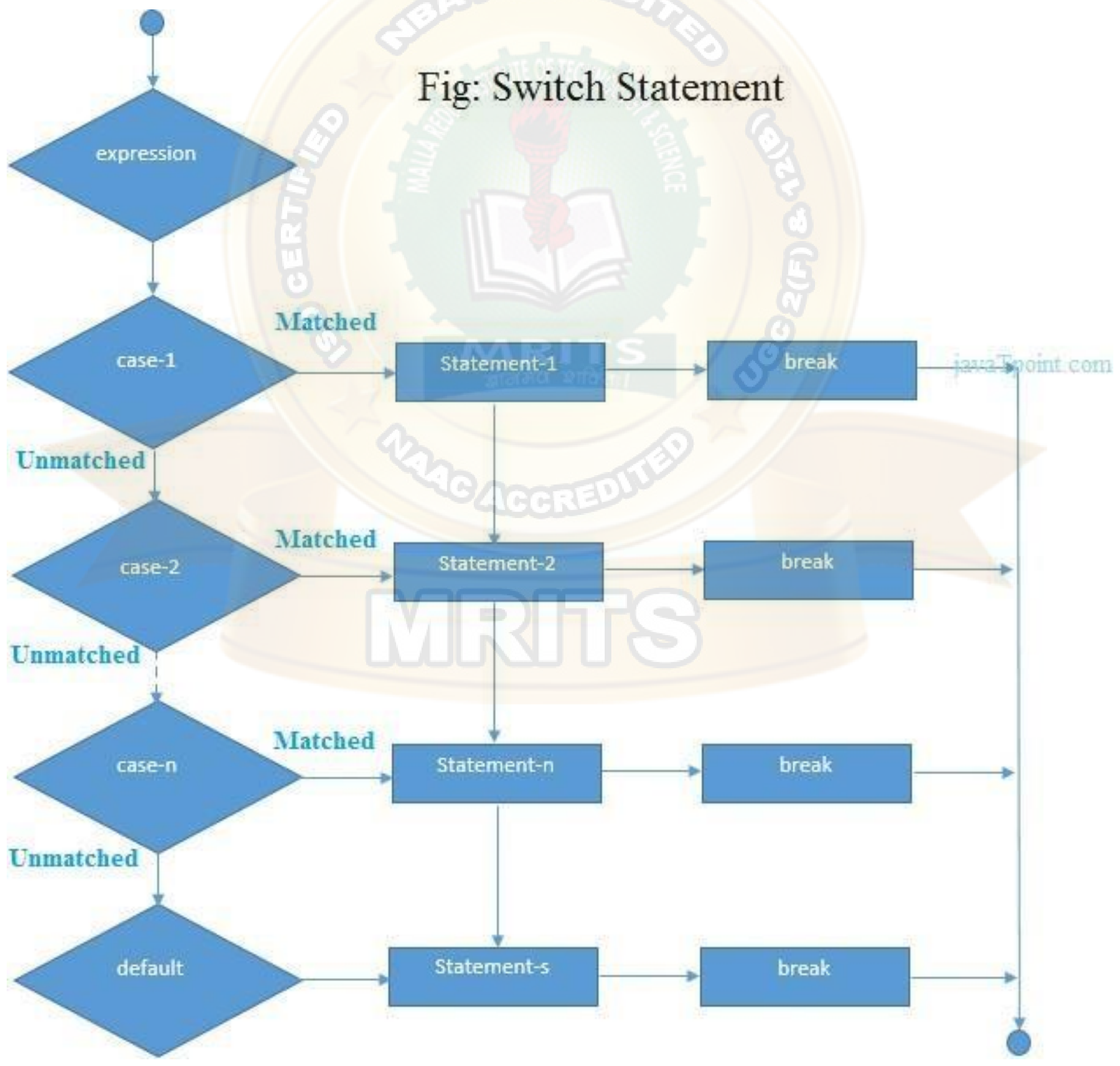
PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

```
switch(expression){
case value1:
 //code to be executed
 break;
case value2:
 //code to be executed
```

break;
......
default:
code to be executed **if** all cases are not matched;
}

PHP Switch Flowchart



PHP Switch Example

```
<?php
$num=20;
switch($num){
case 10:
echo("number is equals to 10");
break;
case 20:
echo("number is equal to 20");
break;
case 30:
echo("number is equal to 30");
break;
default:
echo("number is not equal to 10, 20 or 30");
}
?>
```

Output:

```
number is equal to 20
```

PHP For Loop

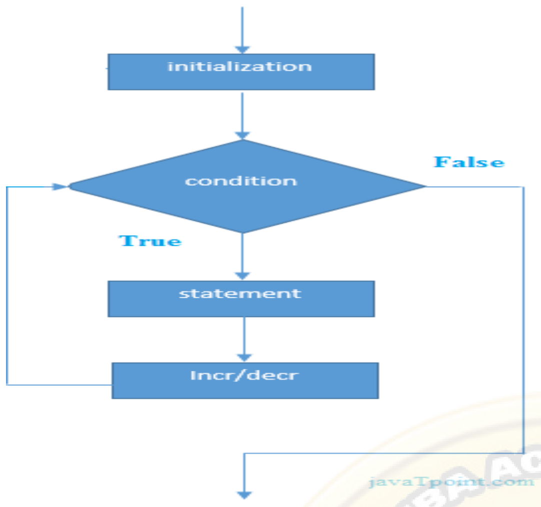
PHP for loop can be used to traverse set of code for the specified number of times.

It should be used if number of iteration is known otherwise use while loop.

Syntax

```
for(initialization; condition; increment/decrement){
//code to be executed
}
```

Flowchart



Example

```
<?php
for($n=1;$n<=10;$n++){
echo "$n<br/>";
}
?>
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop.

In case of inner or nested for loop, nested for loop is executed fully for one outer for loop. If outer for loop is to be executed for 3 times and inner for loop for 3 times, inner for loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example

```
<?php
for($i=1;$i<=3;$i++){
for($j=1;$j<=3;$j++){
echo "$i  $j<br/>";
}
}
?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP For Each Loop

PHP for each loop is used to traverse array elements.

Syntax

```
foreach( $array as $var ){
//code to be executed
}
?>
```

Example

```
<?php
$season=array("summer","winter","spring","autumn");
foreach( $season as $arr ){
echo "Season is: $arr<br />";
}
?>
```

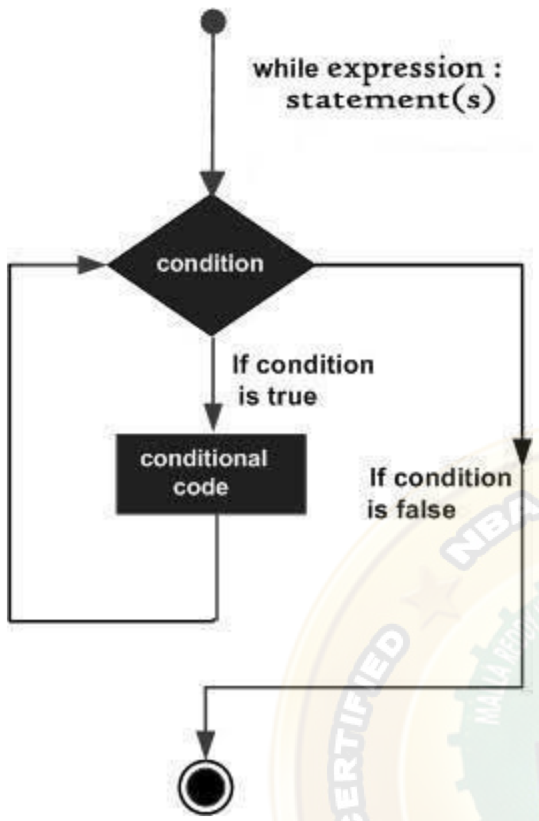
Output:

```
Season is: summer
Season is: winter
Season is: spring
Season is: autumn
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



Syntax
while (*condition*) {
code to be executed;
}

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>

<?php
        $i =0;
        $num =50;

while( $i <10){
           $num--;
           $i++;
}

        echo ("Loop stopped at i = $i and num = $num");
?>
```

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

This will produce the following result −

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

do {
code to be executed;
}
while (*condition*);

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 −

```
<html>
<body>

<?php
        $i =0;
        $num =0;

do{
          $i++;
}

while( $i <10);
        echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

This will produce the following result −

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

foreach (*array* as *value*) {
code to be executed;

}

Example

Try out following example to list out the values of an array.

```
<html>
<body>

<?php
        $array = array(1,2,3,4,5);

foreach( $array as $value ){
           echo "Value is $value <br />";
}
?>

</body>
</html>
```

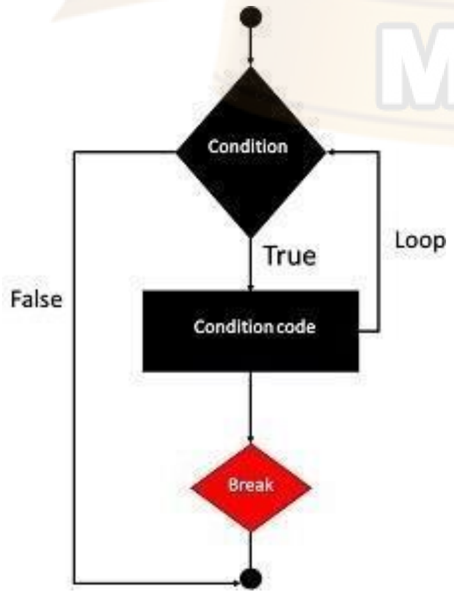
This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>

<?php
        $i =0;

while( $i <10){
          $i++;
if( $i ==3)break;
}
        echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

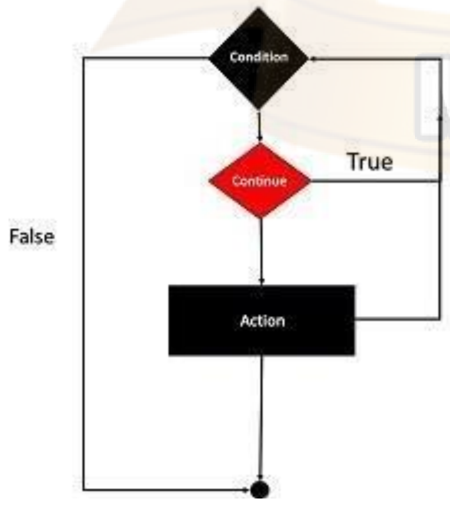
This will produce the following result −

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
<body>

<?php
        $array = array(1,2,3,4,5);

foreach( $array as $value ){
if( $value ==3)continue;
          echo "Value is $value <br />";
}
?>

</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

Advantage of PHP Functions

Code Reusability: PHP functions are defined only once and can be invoked many times, like in other programming languages.

Less Code: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

Easy to understand: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare user-defined functions.

Syntax

```
function functionname(){
//code to be executed
}
```

Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.

PHP Functions Example

File: function1.php

```
<?php
function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

Output:

```
Hello PHP Function
```

PHP Function Arguments

We can pass the information in PHP function through arguments which is separated by comma.

PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and **Variable-length argument list**.

Let's see the example to pass single argument in PHP function.

File: functionarg.php

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
sayHello("Sonoo");
sayHello("Vimal");
sayHello("John");
?>
```

Output:

```
Hello Sonoo
Hello Vimal
Hello John
```

Let's see the example to pass two argument in PHP function.

File: functionarg2.php

```
<?php
function sayHello($name,$age){
echo "Hello $name, you are $age years old<br/>";
}
sayHello("Sonoo",27);
sayHello("Vimal",29);
sayHello("John",23);
?>
```

Output:

```
Hello Sonoo, you are 27 years old
Hello Vimal, you are 29 years old
Hello John, you are 23 years old
```

PHP Call By Reference

Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Let's see a simple example of call by reference in PHP.

File: functionref.php

```
<?php
function adder(&$str2)
{
$str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

Output:

```
Hello Call By Reference
```

PHP Function: Default Argument Value

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

File: functiondefaultarg.php

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

Output:

```
Hello Rajesh
Hello  Sonoo
Hello John
```

PHP Function: Returning Value

Let's see an example of PHP function that returns value.

File: functiondefaultarg.php

```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
Output:
```

```
Cube of 3 is: 27
```

Reading data from the Web.

PHP Form Handling

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $_GET and $_POST.

The form request may be get or post. To retrieve data from get request, we need to use $_GET, for post request $_POST.

PHP Get Form

Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.

Let's see a simple example to receive data from get request in PHP.

File: form1.html

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name"/>
<input type="submit" value="visit"/>
</form>
```

File: welcome.php

```
<?php
$name=$_GET["name"];//receiving name field value in $name variable
echo "Welcome, $name";
?>
```

PHP Post Form

Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.

The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

Let's see a simple example to receive data from post request in PHP.

File: form1.html

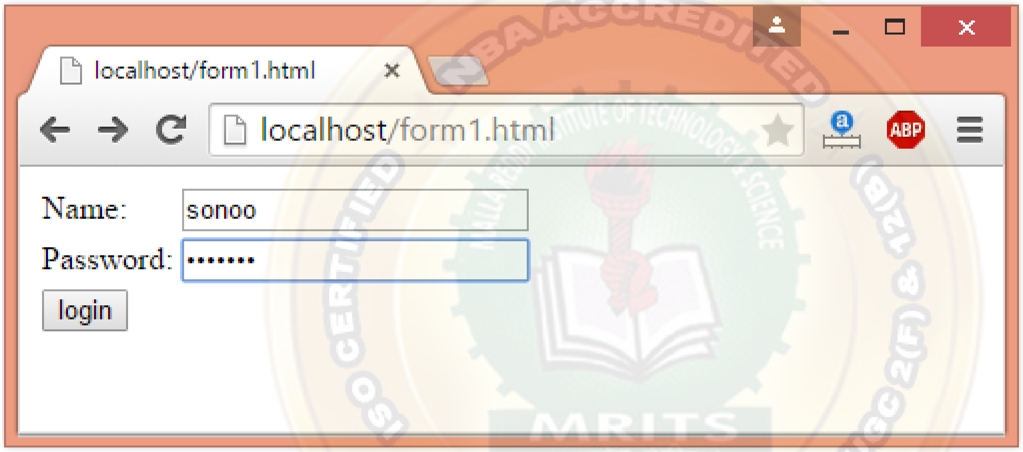
```
<form action="login.php" method="post">
<table>
<tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
```

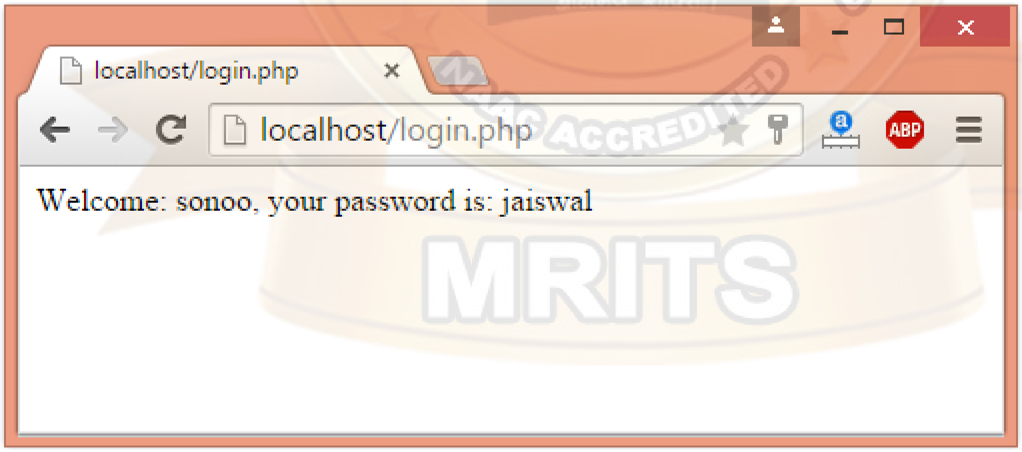
```
    <tr><td>Password:</td><td> <input type="password" name="password"/></td></tr>
    <tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>
    </table>
    </form>
```

File: login.php

```
    <?php
    $name=$_POST["name"];//receiving name field value in $name variable
    $password=$_POST["password"];//receiving password field value in $password variable ec
    ho "Welcome: $name, your password is: $password";
?>
```

Output:





Welcome: sonoo, your password is: jaiswal

Using HTML Forms

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

Try out following example by putting the source code in test.php script.

```
<?php
if( $_POST["name"]|| $_POST["age"]){
```

```
if(preg_match("/[^A-Za-z'-]/",$_POST['name'])){
die("invalid name and name should be alpha");
}

    echo "Welcome ". $_POST['name']."<br />";
    echo "You are ". $_POST['age']." years old.";

exit();
}
?>
<html>
<body>

<form action = "<?php $_PHP_SELF?>" method = "POST">
        Name: <inputtype="text"name="name"/>
        Age: <inputtype="text"name="age"/>
<inputtype="submit"/>
</form>

</body>
</html>
```

It will produce the following result −

- The PHP default variable **$_PHP_SELF** is used for the PHP script name and when you click "submit" button then same PHP script will be called and will produce following result −
- The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in PHP GET & POST chapter.

Browser Redirection

The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

Following example demonstrates how you can redirect a browser request to another web page. Try out this example by putting the source code in test.php script.

```
<?php
if( $_POST["location"]){
    $location = $_POST["location"];
    header("Location:$location");
exit();
}
?>
<html>
<body>

<p>Choose a site to visit :</p>

<form action = "<?php $_SERVER['PHP_SELF']?>" method ="POST">
<selectname="location">.

<optionvalue="http://www.tutorialspoint.com">
            Tutorialspoint.com
</option>

<optionvalue="http://www.google.com">
            Google Search Page
</option>

</select>
<inputtype="submit"/>
</form>

</body>
</html>
```

It will produce the following result −

Displaying "File Download" Dialog Box

Sometime it is desired that you want to give option where a use will click a link and it will pop up a "File Download" box to the user in stead of displaying actual content. This is very easy and will be achieved through HTTP header.

The HTTP header will be different from the actual header where we send **Content-Type** as **text/html\n\n**. In this case content type will be **application/octet-stream** and actual file name will be concatenated along with it.

For example,if you want make a **FileName** file downloadable from a given link then its syntax will be as follows.

```
#!/usr/bin/perl

# HTTP Header
print"Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print"Content-Disposition: attachment; filename=\"FileName\"\r\n\n";
```

```
# Actual File Content
open( FILE,"<FileName");

while(read(FILE, $buffer,100)){
print("$buffer");
}
```

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_GET["name"] || $_GET["age"] ) {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";

    exit();
  }
?>
<html>
<body>
```

```
<form action = "<?php $_PHP_SELF ?>" method = "GET">
        Name: <input type = "text" name = "name" />
        Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>

</body>
</html>
```

It will produce the following result −

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
     if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
        die ("invalid name and name should be alpha");
     }
     echo "Welcome ". $_POST['name']. "<br />";
     echo "You are ". $_POST['age']. " years old.";

     exit();
  }
?>
<html>
<body>

<form action = "<?php $_PHP_SELF ?>" method = "POST">
        Name: <input type = "text" name = "name" />
        Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>
```

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

It will produce the following result −

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_REQUEST["name"] || $_REQUEST["age"] ) {
    echo "Welcome ". $_REQUEST['name']. "<br />";
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
  }
?>
<html>
<body>

<form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>

</body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</a><br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
<body>

<?php include("menu.php"); ?>
<p>This is an example to show how to include PHP file!</p>

</body>
</html>
```

It will produce the following result −

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>

</body>
</html>
```

This will produce the following result −

```
This is an exam<html>
<body>

<?php ple to show how to include wrong PHP file!
```

Now lets try same example with require() function.

```
<html>
<body>

<?php require("xxmenu.php"); ?>
```

```
<p>This is an example to show how to include wrong PHP file!</p>

</body>
</html>
```

Handling file uploads

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps −

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

Creating an upload form

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

```
<?php
if(isset($_FILES['image'])){
   $errors= array();
   $file_name = $_FILES['image']['name'];
   $file_size =$_FILES['image']['size'];
   $file_tmp =$_FILES['image']['tmp_name'];
   $file_type=$_FILES['image']['type'];
   $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

   $expensions= array("jpeg","jpg","png");

if(in_array($file_ext,$expensions)===false){
     $errors[]="extension not allowed, please choose a JPEG or PNG file.";
}
if($file_size >2097152){
        $errors[]='File size must be excately 2 MB';
}

if(empty($errors)==true){
        move_uploaded_file($file_tmp,"images/".$file_name);
        echo "Success";
}else{
        print_r($errors);
}
}
?>
<html>
<body>

<formaction=""method="POST"enctype="multipart/form-data">
<inputtype="file"name="image"/>
<inputtype="submit"/>
</form>

</body>
</html>
```

It will produce the following result −

Creating an upload script

There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables −

- **$_FILES['file']['tmp_name']** − the uploaded file in the temporary directory on the web server.
- **$_FILES['file']['name']** − the actual name of the uploaded file.
- **$_FILES['file']['size']** − the size in bytes of the uploaded file.
- **$_FILES['file']['type']** − the MIME type of the uploaded file.
- **$_FILES['file']['error']** − the error code associated with this file upload.

Example

Below example should allow upload images and gives back result as uploaded file information.

```
<?php
if(isset($_FILES['image'])){
      $errors= array();
      $file_name = $_FILES['image']['name'];
      $file_size = $_FILES['image']['size'];
      $file_tmp = $_FILES['image']['tmp_name'];
      $file_type = $_FILES['image']['type'];
      $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

      $expensions= array("jpeg","jpg","png");

if(in_array($file_ext,$expensions)===false){
        $errors[]="extension not allowed, please choose a JPEG or PNG file.";
}

if($file_size >2097152){
        $errors[]='File size must be excately 2 MB';
}

if(empty($errors)==true){
        move_uploaded_file($file_tmp,"images/".$file_name);
        echo "Success";
}else{
        print_r($errors);
}
}
?>
<html>
<body>

<formaction=""method="POST"enctype="multipart/form-data">
<inputtype="file"name="image"/>
<inputtype="submit"/>

<ul>
<li>Sent file: <?php echo $_FILES['image']['name'];?>
<li>File size: <?php echo $_FILES['image']['size'];?>
<li>File type: <?php echo $_FILES['image']['type']?>
</ul>

</form>
</body>
</html>
```

Every company follows a different coding standard based on their best practices. Coding standard is required because there may be many developers working on different modules so if they will start inventing their own standards then source will become very un-manageable and it will become difficult to maintain that source code in future.

Here are several reasons why to use coding specifications −

- Your peer programmers have to understand the code you produce. A coding standard acts as the blueprint for all the team to decipher the code.
- Simplicity and clarity achieved by consistent coding saves you from common mistakes.
- If you revise your code after some time then it becomes easy to understand that code.
- Its industry standard to follow a particular standard to being more quality in software.

There are few guidelines which can be followed while coding in PHP.

- **Indenting and Line Length** − Use an indent of 4 spaces and don't use any tab because different computers use different setting for tab. It is recommended to keep lines at approximately 75-85 characters long for better code readability.
- **Control Structures** − These include if, for, while, switch, etc. Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls. You are strongly encouraged to always use curly braces even in situations where they are technically optional.

Examples

```
if ((condition1) || (condition2)) {
  action1;
}elseif ((condition3) && (condition4)) {
  action2;
}else {
  default action;
}
```

You can write switch statements as follows −

```
switch (condition) {
  case 1:
    action1;
    break;

  case 2:
    action2;
    break;

  default:
    defaultaction;
    break;
}
```

- **Function Calls** − Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example −

```
$var = foo($bar, $baz, $quux);
```

- **Function Definitions** − Function declarations follow the "BSD/Allman style" −

```
function fooFunction($arg1, $arg2 = '') {
  if (condition) {
    statement;
  }
  return $val;
}
```

- **Comments** − C style comments (/* */) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.
- **PHP Code Tags** − Always use <?php ?> to delimit PHP code, not the <? ?> shorthand. This is required for PHP compliance and is also the most portable way to include PHP code on differing operating systems and setups.
- **Variable Names** −
    - Use all lower case letters
    - Use '_' as the word separator.
    - Global variables should be prepended with a 'g'.
    - Global constants should be all caps with '_' separators.
    - Static variables may be prepended with 's'.
- **Make Functions Reentrant** − Functions should not keep static variables that prevent a function from being reentrant.
- **Alignment of Declaration Blocks** − Block of declarations should be aligned.
- **One Statement Per Line** − There should be only one statement per line unless the statements are very closely related.
- **Short Methods or Functions** − Methods should limit themselves to a single page of code.

Connecting to Database (MYSQL)

PHP MySQL Connect

Since PHP 5.5, **mysql_connect()** extension is *deprecated*. Now it is recommended to use one of the 2 alternatives.

mysqli_connect()

PDO::_construct()

PHP mysqli_connect()

PHP **mysqli_connect() function** is used to connect with MySQL database. It returns *resource* if connection is established or *null*.

Syntax

resource mysqli_connect (server, username, password)

PHP mysqli_close()

PHP **mysqli_close() function** is used to disconnect with MySQL database. It returns *true* if connection is closed or *false*.

Syntax

```
bool mysqli_close(resource $resource_link)
```

PHP MySQL Connect Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$conn = mysqli_connect($host, $user, $pass);
if(! $conn )
{
  die('Could not connect: ' . mysqli_error());
}
echo 'Connected successfully';
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
```

PHP MySQL Create Database

Since PHP 4.3, **mysql_create_db()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

mysqli_query()

PDO::_query()

PHP MySQLi Create Database Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$conn = mysqli_connect($host, $user, $pass);
if(! $conn )
{
  die('Could not connect: ' . mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = 'CREATE Database mydb';
if(mysqli_query( $conn,$sql)){
  echo "Database mydb created successfully.";
}else{
echo "Sorry, database creation failed ".mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Database mydb created successfully.
```

PHP MySQL Create Table

PHP mysql_query() function is used to create table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

mysqli_query()

PDO::_query()

PHP MySQLi Create Table Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = "create table emp5(id INT AUTO_INCREMENT,name VARCHAR(20) NOT NULL
,
emp_salary INT NOT NULL,primary key (id))";
if(mysqli_query($conn, $sql)){
 echo "Table emp5 created successfully";
}else{
echo "Could not create table: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Table emp5 created successfully
```

PHP MySQL Delete Record

PHP mysql_query() function is used to delete record in a table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

mysqli_query()

PDO::_query()

PHP MySQLi Delete Record Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$id=2;
$sql = "delete from emp4 where id=$id";
if(mysqli_query($conn, $sql)){
 echo "Record deleted successfully";
}else{
echo "Could not deleted record: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Record deleted successfully
```

PHP MySQL Select Query

PHP mysql_query() function is used to execute select query. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

mysqli_query()

PDO::_query()

There are two other MySQLi functions used in select query.

mysqli_num_rows(mysqli_result $result): returns number of rows.

mysqli_fetch_assoc(mysqli_result $result): returns row as an associative array. Each key of the array represents the column name of the table. It return NULL if there are no more rows.

PHP MySQLi Select Query Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';
$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = 'SELECT * FROM emp4';
$retval=mysqli_query($conn, $sql);

if(mysqli_num_rows($retval) > 0){
 while($row = mysqli_fetch_assoc($retval)){
    echo "EMP ID :{$row['id']} <br> ".
         "EMP NAME : {$row['name']} <br> ".
         "EMP SALARY : {$row['salary']} <br> ".
         "--------------------------------<br>";
 } //end of while
}else{
echo "0 results";
}
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
EMP ID :1
EMP NAME : ratan
EMP SALARY : 9000
----------------------------
EMP ID :2
EMP NAME : karan
EMP SALARY : 40000
----------------------------
```

```
EMP ID :3
EMP NAME : jai
EMP SALARY : 90000
------------------------------------
```

Handling Cookies and Sessions in PHP

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this −

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection:       close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.The browser's headers might look something like this −

A PHP script will then have access to the cookie in the environmental variables $_COOKIE or $HTTP_COOKIE_VARS[] which holds all cookie names and values. Above cookie can be accessed using $HTTP_COOKIE_VARS["name"].

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments −

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
  setcookie("name","John Watkin", time()+3600,"/","",0);
  setcookie("age","36", time()+3600,"/","",0);
?>
<html>

<head>
<title>Setting Cookies with PHP</title>
</head>

<body>
<?php echo "Set Cookies"?>
</body>

</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>

<head>
<title>Accessing Cookies with PHP</title>
```

```
</head>

<body>

<?php
        echo $_COOKIE["name"]."<br />";

/* is equivalent to */
        echo $HTTP_COOKIE_VARS["name"]."<br />";

        echo $_COOKIE["age"]."<br />";

/* is equivalent to */
        echo $HTTP_COOKIE_VARS["name"]."<br />";
?>

</body>
</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>

<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>

<?php
if( isset($_COOKIE["name"]))
          echo "Welcome ". $_COOKIE["name"]."<br />";

else
          echo "Sorry... Not recognized"."<br />";
?>

</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired −

```
<?php
```

```
   setcookie("name","", time()-60,"/","",0);
   setcookie("age","", time()-60,"/","",0);
?>
<html>

<head>
<title>Deleting Cookies with PHP</title>
</head>

<body>
<?php echo "Deleted Cookies"?>
</body>

</html>
```

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen −

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.
Put this code in a test.php file and load this file many times to see the result −

```
<?php
  session_start();

if( isset( $_SESSION['counter'])){
     $_SESSION['counter']+=1;
}else{
     $_SESSION['counter']=1;
}

  $msg ="You have visited this page ". $_SESSION['counter'];
  $msg .="in this session.";
?>

<html>

<head>
<title>Setting up a PHP session</title>
</head>

<body>
<?php echo ( $msg );?>
</body>

</html>
```

It will produce the following result −

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.
Here is the example to unset a single variable −

```
<?php
  unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables −

```
<?php
  session_destroy();
?>
```

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
  session_start();

if(isset($_SESSION['counter'])){
    $_SESSION['counter']=1;
}else{
    $_SESSION['counter']++;
}

  $msg ="You have visited this page ". $_SESSION['counter'];
  $msg .="in this session.";

  echo ( $msg );
?>

<p>
  To continue click following link <br/>

<a href = "nextpage.php?<?php echo htmlspecialchars(SID);?>">
</p>
```

File handling in PHP

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Mode	**Purpose**
R	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
W	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
A	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.
After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.
The files length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.
So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.
<html>

```
<head>
<title>Reading a file using PHP</title>
</head>

<body>

<?php
      $filename = "tmp.txt";
      $file = fopen( $filename, "r" );

      if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
      }

      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );

      echo ( "File size : $filesize bytes" );
      echo ( "<pre>$filetext</pre>" );
    ?>

</body>
</html>
```

Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exist()** function which takes file name as an argument

```
<?php
  $filename = "/home/user/guest/newfile.txt";
  $file = fopen( $filename, "w" );

  if( $file == false ) {
    echo ( "Error in opening new file" );
```

```
      exit();
   }
   fwrite( $file, "This is a simple test\n" );
   fclose( $file );
?>
<html>

<head>
<title>Writing a file using PHP</title>
</head>

<body>

<?php
         $filename = "newfile.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );

         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "$filetext" );
         echo("file name: $filename");
      ?>

</body>
</html>
```

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.
You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.
There are two parts which should be clear to you −

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Directory Functions

Function	Description
chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle
readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

15. Additional Topics

IntroductiontoWorldWideWeb
BasicHTMLtags, Lists,Images,Tables
Forms,Frames
Cascadingstylesheets

Introduction to the World Wide Web

Simple use of the Web

From your home in the United Kingdom, in the small English town of Chipping Sodbury, you decide to find out what entertainment there is in Bath, the famous city only 15 miles away. A company publishes*What's On in Bath* online, and you have read that this is available on http://www.bath.info.uk/ as a series of Web `pages' on the Internet. You decide to take a look. You switch on your computer and start it up in the usual way. Just as you might click on an icon to start up your desktop publishing package, so you click on the icon to load your World Wide Web browser. The browser is a piece of software that allows you to display certain kinds of information from the Internet.

http://www.bath.avon.uk/

HTTP (HyperText Transfer Protocol) is the name of the Web's own transmission protocol. Web pages are sent over the Internet to your computer courtesy of HTTP.

The difference between the Web and the Internet

Do not make the rather common mistake of confusing the Web with the Internet itself: the Internet simply provides the medium for the Web to run on, just as a telephone line provides the medium for telephone conversations. What the Web does is provide the technology for publishing, sending and obtaining information over the expanse of the Internet. How the Internet actually works may be a matter of interest, but the Web user does not need to know about it in any detail.

This fictitious home page would have perhaps a photo showing the famous Roman Baths on the first page, together with a short paragraph introducing the town, and a menu of icons that call up specific information on Bath's museums, parks, bus-tours and so on. These icons provide hypertext links for you to click on-screen `buttons', enabling you to home-in on the information you want. Hypertext links are in many ways the most important feature of the Web. In the case of our imaginary Web pages, clicking on hypertext buttons mostly displays data that is held on computers somewhere in Bath itself. Sometimes, however, a hypertext link may fetch a file from somewhere quite different on the Internet. If you click on the title of a play taking place in Bath, information about the performing company may be fetched across the Internet from a computer thousands of miles away in the United States. From that point on, you might be able to call up pages that tell you about other plays by the same company. With the Web, you can depart on your own private `tour' of information on a chosen subject whenever the inclination takes you. This is by virtue of the hypertext links that span the globe.
you can see that the Web provides the following:

- A means for publishing online information. The Web enables you to lay out text and graphics of the `pages' on the screen, and to insert titles, photos, captions and so on.
- A means for retrieving online information via hypertext links, at the click of a button. You can also use Web search programs to find the information you want.
- An interface for looking at information retrieved. This is done by virtue of a Web browser.

A Web page may also contain forms for conducting commercial transactions across the Internet and include other applications; for example, spreadsheets, video clips, sound clips and so on.

The Web in the context of the Internet

The Internet is a vast network of interconnected computers. Just as AT&T, France Telecom, British Telecom, and other countrywide or regional telephone networks now are joined together to form a global telephone system, so it is with the Internet. The many thousands of computer networks that make up the Internet are joined together on a global scale, so that any Internet computer can communicate with any other.
How can your computer know how to find someone else's computer on the other side of the world? Just as each telephone in the world has its own unique telephone number, so each

computer connected to the Internet has its own computer number. This is known as its IP, or Internet Protocol, address. However, because IP addresses consist of long series of numbers that are cumbersome to remember and type, you rarely come across them in everyday Internet use; most people prefer to use the parallel system of naming computers. This is the system of Internet host names, sometimes called Internet addresses or even domain names. Whereas a computer on the Internet may have an IP address such as 17.254.0.63, it may have a more manageable Internet host or domain name such as www.drizzle.org. Servers consult a globally distributed directory to map each host name onto the corresponding IP address.

Basic components of the Web

The basic components of the Web are shown in the following illustration. They are:

- **Web servers**, which are computers that hold information for distribution over the Internet. In the example application in the diagram, one Web server might hold the text and graphics of the online magazine *What's On in Bath*, and another server might hold information on which seats are available for a particular concert. The magazine would be formatted using the Web's own publishing language, HTML (HyperText Mark-up Language). The data on available seats and their price would be held in a database with links to specific forms that are published using HTML.
- **Servers**, which can be PCs, Macintosh systems or UNIX workstations: it is the server software that makes them special, rather than the computer itself. That said, servers need to be fairly up-market machines. Servers do need to be left on all the time, so that people can access the information on them whenever they want. Another important point about servers: they are relatively difficult to set up. If you are a non-technical person who wants to publish on the Web, the best thing to do is to rent some space on someone else's server.
- **Web clients**, which can be PCs, Macintoshes and other computers that are connected to the Internet and which can retrieve information from Web servers. A Web client is the computer on your desk. PCs, Macintoshes, UNIX workstations and even simple terminals can run client software. Different client software is marketed (or is given away free) for different platforms. Thus, Mosaic has both a Macintosh and a PC implementation.
- **HTTP protocol**, which is used to transmit files between servers and clients. When you click on a hypertext link or fill out a form in a Web document, the results need to be sent across the Internet as quickly as possible, and then to be understood by a server at the other end. Instructions such as `send me this file' or `get me that image' are carried by the Web communications protocol, HTTP. This protocol is the `messenger' that fetches files to and from servers, and then delivers results to your computer every time you click with a request. HTTP has its counterparts in other Internet services: FTP, file transfer protocol, and Gopher are protocols that obtain different sorts of information from across the Internet.

- **Browser** software, which is needed by a Web client for displaying text, images, video clips and so on. This is supplied under the umbrella name `browser', of which Mosaic, Microsoft Corp.'s Internet Explorer and Netscape Communications Corp.'s Navigator and Communicator browsers are probably the best-known examples. Browser software gives you the ability to scan information retrieved from Web servers, as you would browse through a book. It also gives you facilities for saving and printing information obtained on the Web.

A universally understood publishing language: HTML

To publish information for global distribution on the World Wide Web, you need a universally understood publishing language, a kind of mother tongue, which all computers on the Web can potentially understand. You also need a commonly understood communications protocol for sending published information `down the wire' from computer to computer. This should enable users to download information to their machine at the click of a button, and also to send back information (your address, a credit card number, a query to a database and so on) with little effort.

The publishing language used by the Web is called HTML (HyperText Mark-up Language). Using HTML, you can specify which parts of your text are to be headings, paragraphs, bulleted lists, and which parts are to be rendered in bold-face type, in italicized type and so on. You can use HTML to insert tables into documents, to write equations, to import images and to format fill-out forms for querying databases at a distance. (Some of these features are specific to HTML 3 and are not supported by earlier versions of HTML.) The HTML language itself is very flexible and not difficult to use, although, as with all tasks associated with computing, patience is a necessary virtue for authoring hypertext. Part of the Web's appeal is that almost anyone with a reasonable PC, Macintosh or UNIX computer can publish information without being unduly technical. Judging by the variety of publishers on the Web today, HTML is within the grasp of many. A simple example of HTML can be seen at the beginning of Chapter 3.

The HTTP protocol

[FIGURE in margin -- long distance jump p.9]

The initial protocol for the Web was very, very simple. The client sent a request: `GET this filename' and the other end sent back the file and closed the connection. And that was it. There was no content type to tell you what kind of file was being sent. No status code. Just the file. The client therefore had to guess what it had been given and this developed into a fine art. First, the browser would look at the file extension to see if there were any clues, such as ˙GIF or ˙HTML, and then it would look at the beginning of the file in case the first few bytes gave the game away - all rather precarious.

HTML-Overview

HTML stands for **H**yper**t**ext **M**arkup **L**anguage, and it is the most widely used language to write Web Pages.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.

- As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

BasicHTMLDocument

In its simplest form, following is an example of an HTML document −

```
<!DOCTYPE html>
<html>
<head>
<title>This is document title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>Document content goes here ....</p>
</body>
</html>
```

HTMLTags

As told earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within angle braces **<Tag Name>**. Except few tags, most of the tags have their corresponding closing tags. For example, **<html>** has its closing tag **</html>** and **<body>** tag has its closing tag **</body>** tag etc.

Above example of HTML document uses the following tags −

Sr.No	Tag & Description
1	**<!DOCTYPE...>** This tag defines the document type and HTML version.
2	**<html>** This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
3	**<head>** This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.

4	**<title>** The <title> tag is used inside the <head> tag to mention the document title.
5	**<body>** This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.
6	**<h1>** This tag represents the heading.
7	**<p>** This tag represents a paragraph.

To learn HTML, you will need to study various tags and understand how they behave, while formatting a textual document. Learning HTML is simple as users have to learn the usage of different tags in order to format the text or images to make a beautiful webpage.

World Wide Web Consortium (W3C) recommends to use lowercase tags starting from HTML 4.

HTMLDocumentStructure

A typical HTML document will have the following structure −

```
<html>
<head>
    Document header related tags
</head>
<body>
    Document body related tags
</body>
</html>
```

We will study all the header and body tags in subsequent chapters, but for now let's see what is document declaration tag.

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>,** and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Heading Example</title>
</head>
<body>
```

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
</body>
</html>
```

This will produce the following result −

ParagraphTag

The **<p>** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening <p> and a closing </p> tag as shown below in the example −

LineBreak Tag

Whenever you use the **<br />** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The <br /> tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use <br> it is not valid in XHTML.

HorizontalLines

Horizontal lines are used to visually break-up sections of a document. The **<hr>** tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

For example, you may want to give a line between two paragraphs as in the given example below −

PreserveFormatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag **<pre>**.

Any text between the opening **<pre>** tag and the closing **</pre>** tag will preserve the formatting of the source document.

HTML-Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells. The elements under <td> are regular and left aligned by default

CellpaddingandCellspacingAttributes

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells. The cellspacing attribute defines space between table cells, while cellpadding represents the distance between cell borders and the content within a cell.

ColspanandRowspanAttributes

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

TablesBackgrounds

You can set table background using one of the following two ways −

- **bgcolor** attribute − You can set background color for whole table or just for one cell.
- **background** attribute − You can set background image for whole table or just for one cell.

You can also set border color also using **bordercolor** attribute.

Note − The *bgcolor*, *background*, and *bordercolor* attributes deprecated in HTML5. Do not use these attributes.

TableHeightandWidth

You can set a table width and height using **width** and **height** attributes. You can specify table width or height in terms of pixels or in terms of percentage of available screen area.

The **caption** tag will serve as a title or explanation for the table and it shows up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

TableHeader,Body,andFooter

Tables can be divided into three portions − a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are −

- **<thead>** − to create a separate table header.
- **<tbody>** − to indicate the main body of the table.
- **<tfoot>** − to create a separate table footer.

A table may contain several <tbody> elements to indicate *different pages* or groups of data. But it is notable that <thead> and <tfoot> tags should appear before <tbody>

NestedTables

You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag <td>.

HTML–Lists

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain −

- **<ul>** − An unordered list. This will list items using plain bullets.
- **<ol>** − An ordered list. This will use different schemes of numbers to list your items.
- **<dl>** − A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTMLUnorderedLists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML **<ul>** tag. Each item in the list is marked with a bullet.

HTMLOrderedLists

If you are required to put your items in a numbered list instead of bulleted, then HTML ordered list will be used. This list is created by using **<ol>** tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with <li>.

HTMLDefinitionLists

HTML and XHTML supports a list style which is called **definition lists** where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

- <dl> − Defines the start of the list
- <dt> − A term
- <dd> − Term definition
- </dl> − Defines the end of the list

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.

DisadvantagesofFrames

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.

Note − The <frame> tag deprecated in HTML5. Do not use this element.

Example

Following is the example to create three horizontal frames −

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
```

```
<framesetrows="10%,80%,10%">
<framename="top"src="/html/top_frame.htm"/>
<framename="main"src="/html/main_frame.htm"/>
<framename="bottom"src="/html/bottom_frame.htm"/>
<noframes>
<body>Your browser does not support frames.</body>
</noframes>
</frameset>
</html>
```

The<frameset>TagAttributes

Following are important attributes of the <frameset> tag −

Sr.No	Attribute & Description
1	**cols** Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways − Absolute values in pixels. For example, to create three vertical frames, use *cols = "100, 500, 100"*. A percentage of the browser window. For example, to create three vertical frames, use *cols = "10%, 80%, 10%"*. Using a wildcard symbol. For example, to create three vertical frames, use *cols = "10%, *, 10%"*. In this case wildcard takes remainder of the window. As relative widths of the browser window. For example, to create three vertical frames, use *cols = "3*, 2*, 1*"*. This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.
2	**rows** This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use *rows = "10%, 90%"*. You can specify the height of each row in the same way as explained above for columns.
3	**border** This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.
4	**frameborder** This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example

	frameborder = "0" specifies no border.
5	**framespacing** This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing = "10" means there should be 10 pixels spacing between each frames.

The<frame>TagAttributes

Following are the important attributes of <frame> tag −

Sr.No	Attribute & Description
1	**src** This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.
2	**name** This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
3	**frameborder** This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
4	**marginwidth** This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth = "10".
5	**marginheight** This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight = "10".
6	**noresize** By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize = "noresize".

7	**scrolling** This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling = "no" means it should not have scroll bars.
8	**longdesc** This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc = "framedescription.htm"

HTML-Forms

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax −

```
<form action = "Script URL" method = "GET|POST">
  form elements like input, textarea etc.
</form>
```

FormAttributes

Apart from common attributes, following is a list of the most frequently used form attributes −

Sr.No	Attribute & Description
1	**action** Backend script ready to process your passed data.
2	**method** Method to be used to upload data. The most frequently used are GET and POST methods.
3	**target** Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
4	**enctype** You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are −

	application/x-www-form-urlencoded − This is the standard method most forms use in simple scenarios. **mutlipart/form-data** − This is used when you want to upload binary data in the form of files like image, word file etc.

Note − You can refer to Perl & CGI for a detail on how form data upload works.

HTMLForm Controls

There are different types of form controls that you can use to collect data using HTML form −

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

TextInputControls

There are three types of text input used on forms −

- **Single-line text input controls** − This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML **<input>** tag.
- **Password input controls** − This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTMl <input> tag.
- **Multi-line text input controls** − This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML **<textarea>** tag.

Single-linetextinputcontrols

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Example

Here is a basic example of a single-line text input used to take first name and last name −

```
<!DOCTYPE html>
<html>
<head>
<title>Text Input Control</title>
</head>
<body>
<form>
        First name: <inputtype="text"name="first_name"/>
<br>
        Last name: <inputtype="text"name="last_name"/>
```

```
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating text field.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for text input control it will be set to **text**.
2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value** This can be used to provide an initial value inside the control.
4	**size** Allows to specify the width of the text-input control in terms of characters.
5	**maxlength** Allows to specify the maximum number of characters a user can enter into the text box.

Passwordinputcontrols

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input>tag but type attribute is set to **password**.

Example

Here is a basic example of a single-line password input used to take user password −

```
<!DOCTYPE html>
<html>
<head>
<title>Password Input Control</title>
</head>
<body>
<form>
        User ID : <inputtype="text"name="user_id"/>
<br>
        Password: <inputtype="password"name="password"/>
</form>
</body>
```

```
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating password field.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for password input control it will be set to **password**.
2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value** This can be used to provide an initial value inside the control.
4	**size** Allows to specify the width of the text-input control in terms of characters.
5	**maxlength** Allows to specify the maximum number of characters a user can enter into the text box.

Multiple-LineTextInputControls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Example

Here is a basic example of a multi-line text input used to take item description −

```
<!DOCTYPE html>
<html>
<head>
<title>Multiple-Line Input Control</title>
</head>
<body>
<form>
      Description : <br/>
<textarearows="5"cols="50"name="description">
        Enter description here...
</textarea>
</form>
</body>
```

```
</html>
```

Attributes

Following is the list of attributes for <textarea> tag.

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**rows** Indicates the number of rows of text area box.
3	**cols** Indicates the number of columns of text area box

CheckboxControl

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox.**.

Example

Here is an example HTML code for a form with two checkboxes −

```
<!DOCTYPE html>
<html>
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<inputtype="checkbox"name="maths"value="on"> Maths
<inputtype="checkbox"name="physics"value="on"> Physics
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for <checkbox> tag.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for checkbox input control it will be set to **checkbox.**.

2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value** The value that will be used if the checkbox is selected.
4	**checked** Set to *checked* if you want to select it by default.

RadioButtonControl

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **radio**.

Example

Here is example HTML code for a form with two radio buttons −

```
<!DOCTYPE html>
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
<inputtype="radio"name="subject"value="maths"> Maths
<inputtype="radio"name="subject"value="physics"> Physics
</form>
</body>
</html>
```

This will produce the following result −

Attributes

Following is the list of attributes for radio button.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for checkbox input control it will be set to radio.
2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value**

	The value that will be used if the radio box is selected.
4	**checked** Set to *checked* if you want to select it by default.

SelectBoxControl

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Example

Here is example HTML code for a form with one drop down box

```
<!DOCTYPE html>
<html>
<head>
<title>Select Box Control</title>
</head>

<body>
<form>
<selectname="dropdown">
<optionvalue="Maths"selected>Maths</option>
<optionvalue="Physics">Physics</option>
</select>
</form>
</body>
</html>
```

Attributes

Following is the list of important attributes of <select> tag −

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**size** This can be used to present a scrolling list box.
3	**multiple** If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag −

Sr.No	Attribute & Description

1	**Value** The value that will be used if an option in the select box box is selected.
2	**selected** Specifies that this option should be the initially selected value when the page loads.
3	**label** An alternative way of labeling options

FileUploadBox

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

Example

Here is example HTML code for a form with one file upload box −

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<inputtype="file"name="fileupload"accept="image/*"/>
</form>
</body>
</html>
```

Attributes

Following is the list of important attributes of file upload box −

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**accept** Specifies the types of files that the server accepts.

ButtonControls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input>tag by setting its type attribute to **button**. The type attribute can take the following values −

Sr.No	Type & Description
1	**submit** This creates a button that automatically submits a form.
2	**reset** This creates a button that automatically resets form controls to their initial values.
3	**button** This creates a button that is used to trigger a client-side script when the user clicks that button.
4	**image** This creates a clickable button but we can use an image as background of the button.

Example

Here is example HTML code for a form with three types of buttons −

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<inputtype="submit"name="submit"value="Submit"/>
<inputtype="reset"name="reset"value="Reset"/>
<inputtype="button"name="ok"value="OK"/>
<inputtype="image"name="imagebutton"src="/html/images/logo.png"/>
</form>
</body>
</html>
```

HiddenFormControls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed current page.

Example

Here is example HTML code to show the usage of hidden control −

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<p>This is page 10</p>
<inputtype="hidden"name="pagename"value="10"/>
<inputtype="submit"name="submit"value="Submit"/>
<inputtype="reset"name="reset"value="Reset"/>
</form>
</body>
</html>
```

CSS

CSS tutorial or CSS 3 tutorial provides basic and advanced concepts of CSS technology. Our CSS tutorial is developed for beginners and professionals. The major points of CSS are given below:

- CSS stands for Cascading Style Sheet.
- CSS is used to design HTML tags.
- CSS is a widely used language on the web.
- HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

CSS Example with CSS Editor

In this tutorial, you will get a lot of CSS examples, you can edit and run these examples with our online CSS editor tool.

```
<!DOCTYPE>
    <html>
    <head>
    <style>
    h1{
    color:white;
    background-color:red;
    padding:5px;
    }
    p{
    color:blue;
    }
    </style>
    </head>
```

```
<body>
<h1>Write Your First CSS Example</h1>
<p>This is Paragraph.</p>
</body>
</html>
```

Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of style sheets so it is advised to use this method sparingly.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

Syntax:

```
<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>
```

Example:

```
<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>
<p>This paragraph is not affected.</p>
```

Output:

Inline CSS is applied on this heading.

This paragraph is not affected.

Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in <head> section of the HTML page inside the <style> tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
background-color: linen;
}
h1 {
color: red;
margin-left: 80px;
}
```

```
</style>
</head>
<body>
<h1>The internal style sheet is applied on this heading.</h1>
<p>This paragraph will not be affected.</p>
</body>
</html>
```

External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Example:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.

Let's take an example of a style sheet file named "mystyle.css".

File: mystyle.css

```
body {
background-color: lightblue;
}
h1 {
color: navy;
margin-left: 20px;
}
```

Note: You should not use a space between the property value and the unit. For example: It should be margin-left:20px not margin-left:20 px.

UNIT – II

HTML- Forms

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax −

```
<form action = "Script URL" method = "GET|POST">
   form elements like input, textarea etc.
</form>
```

FormAttributes

Apart from common attributes, following is a list of the most frequently used form attributes −

Sr.No	Attribute & Description
1	**Action** Backend script ready to process your passed data.
2	**Method** Method to be used to upload data. The most frequently used are GET and POST methods.
3	**Target** Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.

4	**enctype** You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are − **application/x-www-form-urlencoded** − This is the standard method most forms use in simple scenarios. **mutlipart/form-data** − This is used when you want to upload binary data in the form of files like image, word file etc.

Note − You can refer to Perl & CGI for a detail on how form data upload works.

HTMLFormControls

There are different types of form controls that you can use to collect data using HTML form −

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Text Input Controls

There are three types of text input used on forms −

- **Single-line text input controls** − This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML **<input>** tag.
- **Password input controls** − This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTMl <input> tag.
- **Multi-line text input controls** − This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML **<textarea>** tag.

Single-linetextinputcontrols

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Example

Here is a basic example of a single-line text input used to take first name and last name −

```
<!DOCTYPE html>
<html>
<head>
<title>Text Input Control</title>
</head>
<body>
<form>
        First name: <inputtype="text"name="first_name"/>
<br>
        Last name: <inputtype="text"name="last_name"/>
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating text field.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for text input control it will be set to **text**.

2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value** This can be used to provide an initial value inside the control.
4	**size** Allows to specify the width of the text-input control in terms of characters.
5	**maxlength** Allows to specify the maximum number of characters a user can enter into the text box.

Passwordinputcontrols

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input>tag but type attribute is set to **password**.

Example

Here is a basic example of a single-line password input used to take user password −

```
<!DOCTYPE html>
<html>
<head>
<title>Password Input Control</title>
</head>
<body>
<form>
        User ID : <inputtype="text"name="user_id"/>
<br>
        Password: <inputtype="password"name="password"/>
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating password field.

Sr.No	Attribute & Description
1	**type**

	Indicates the type of input control and for password input control it will be set to **password**.
2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value** This can be used to provide an initial value inside the control.
4	**size** Allows to specify the width of the text-input control in terms of characters.
5	**maxlength** Allows to specify the maximum number of characters a user can enter into the text box.

Multiple-Line Text Input Controls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Example

Here is a basic example of a multi-line text input used to take item description −

```
<!DOCTYPE html>
<html>
<head>
<title>Multiple-Line Input Control</title>
</head>
<body>
<form>
        Description : <br/>
<textarearows="5"cols="50"name="description">
           Enter description here...
</textarea>
</form>
</body>

</html>
```

Attributes

Following is the list of attributes for <textarea> tag.

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**rows** Indicates the number of rows of text area box.
3	**cols** Indicates the number of columns of text area box

Checkbox Control

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox.**.

Example

Here is an example HTML code for a form with two checkboxes −

```
<!DOCTYPE html>
<html>
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<inputtype="checkbox"name="maths"value="on"> Maths
<inputtype="checkbox"name="physics"value="on"> Physics
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for <checkbox> tag.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for checkbox input control it will be set to **checkbox.**.
2	**name** Used to give a name to the control which is sent to the server to be recognized and

	get the value.
3	**value** The value that will be used if the checkbox is selected.
4	**checked** Set to *checked* if you want to select it by default.

Radio ButtonControl

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **radio**.

Example

Here is example HTML code for a form with two radio buttons −

```
<!DOCTYPE html>
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
<inputtype="radio"name="subject"value="maths"> Maths
<inputtype="radio"name="subject"value="physics"> Physics
</form>
</body>
</html>
```

Attributes

Following is the list of attributes for radio button.

Sr.No	Attribute & Description
1	**type** Indicates the type of input control and for checkbox input control it will be set to radio.
2	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
3	**value**

	The value that will be used if the radio box is selected.
4	**checked** Set to *checked* if you want to select it by default.

Select BoxControl

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Example

Here is example HTML code for a form with one drop down box

```
<!DOCTYPE html>
<html>
<head>
<title>Select Box Control</title>
</head>
<body>
<form>
<selectname="dropdown">
<optionvalue="Maths"selected>Maths</option>
<optionvalue="Physics">Physics</option>
</select>
</form>
</body>
</html>
```

Attributes

Following is the list of important attributes of <select> tag −

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**size** This can be used to present a scrolling list box.
3	**multiple** If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag −

Sr.No	Attribute & Description
1	**value** The value that will be used if an option in the select box box is selected.
2	**selected** Specifies that this option should be the initially selected value when the page loads.
3	**label** An alternative way of labeling options

File UploadBox

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

Example

Here is example HTML code for a form with one file upload box −

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>

<body>
<form>
<inputtype="file"name="fileupload"accept="image/*"/>
</form>
</body>
</html>
```

Attributes

Following is the list of important attributes of file upload box −

Sr.No	Attribute & Description
1	**name** Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**accept**

	Specifies the types of files that the server accepts.

Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input>tag by setting its type attribute to **button**. The type attribute can take the following values −

Sr.No	Type & Description
1	**submit** This creates a button that automatically submits a form.
2	**reset** This creates a button that automatically resets form controls to their initial values.
3	**button** This creates a button that is used to trigger a client-side script when the user clicks that button.
4	**image** This creates a clickable button but we can use an image as background of the button.

Example

Here is example HTML code for a form with three types of buttons −

```
<!DOCTYPE html>
<html>

<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<inputtype="submit"name="submit"value="Submit"/>
<inputtype="reset"name="reset"value="Reset"/>
<inputtype="button"name="ok"value="OK"/>
<inputtype="image"name="imagebutton"src="/html/images/logo.png"/>
</form>
</body>
</html>
```

Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed current page.

Example

Here is example HTML code to show the usage of hidden control −

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<p>This is page 10</p>
<inputtype="hidden"name="pagename"value="10"/>
<inputtype="submit"name="submit"value="Submit"/>
<inputtype="reset"name="reset"value="Reset"/>
</form>
</body>
</html>
```

CSS

CSS tutorial or CSS 3 tutorial provides basic and advanced concepts of CSS technology. Our CSS tutorial is developed for beginners and professionals. The major points of CSS are given below:

- CSS stands for Cascading Style Sheet.
- CSS is used to design HTML tags.
- CSS is a widely used language on the web.
- HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

CSS Example with CSS Editor

In this tutorial, you will get a lot of CSS examples, you can edit and run these examples with our online CSS editor tool.

```
<!DOCTYPE>
    <html>
    <head>
    <style>
```

```
h1{
color:white;
background-color:red;
padding:5px;
}
p{
color:blue;
}
</style>
</head>
<body>
<h1>Write Your First CSS Example</h1>
<p>This is Paragraph.</p>
</body>
</html>
```

Inline CSS

We can apply CSS in a single element by inline CSS technique.
The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of style sheets so it is advised to use this method sparingly.
If you want to use inline CSS, you should use the style attribute to the relevant tag.
Syntax:

```
<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>
```

Example:

```
<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>
<p>This paragraph is not affected.</p>
```

Output:

Inline CSS is applied on this heading.
This paragraph is not affected.

Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in <head> section of the HTML page inside the <style> tag.
Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
background-color: linen;
}
h1 {
color: red;
margin-left: 80px;
}
</style>
</head>
<body>
<h1>The internal style sheet is applied on this heading.</h1>
<p>This paragraph will not be affected.</p>
</body>
</html>
```

External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Example:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.

Let's take an example of a style sheet file named "mystyle.css".

File: mystyle.css

```
body {
background-color: lightblue;
}
h1 {
color: navy;
margin-left: 20px;
}
```

Note: You should not use a space between the property value and the unit. For example: It should be margin-left:20px not margin-left:20 px.

Introduction to XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So *what exactly is a markup language?* Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
<text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instructs computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

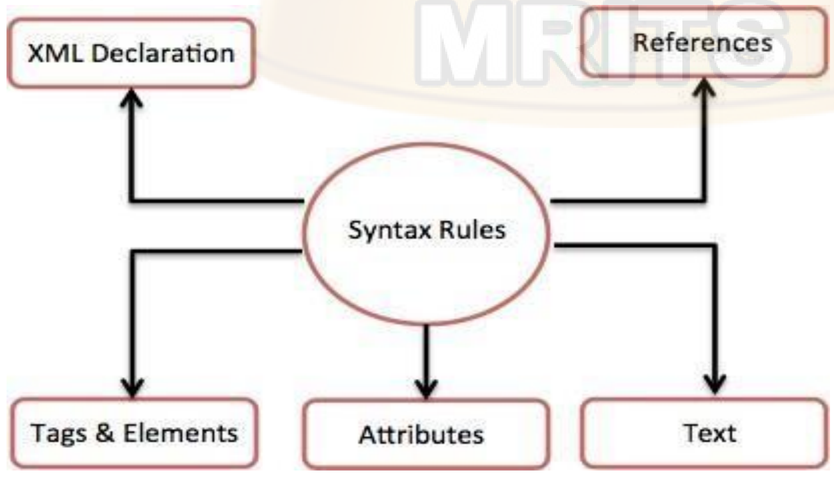
This chapter takes you through the simple syntax rules to write an XML document. Following is a complete XML document:

```
<?xml version="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

You can notice there are two kinds of information in the above example:

- markup, like *<contact-info>* and
- the text, or the character data, *Tutorials Point* and *(040) 123-4567*.

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



Let us see each component of the above diagram in detail:

XML Declaration

The XML document can optionally have an XML declaration. It is written as below:

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

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

Syntax Rules for XML declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets <> as shown below:

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element> ...</element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint
<contact-info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint</company>
<contact-info>
```

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x> ..</x>
```

```
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
<x>...</x>
<y>...</y></root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example **<contact-info>** is different from **<Contact-Info>**.

Attributes

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example:

<ahref="http://www.tutorialspoint.com/">Tutorialspoint!</a>

Here *href* is the attribute name and *http://www.tutorialspoint.com/* is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice:
  <ab="x"c="y"b="z"> ...</a>
- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax:
  <ab=x>... </a>
  In the above syntax, the attribute value is not defined in quotation marks.

XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol **"&"** ,which is a reserved character and end with the symbol **";"**. XML has two types of references:

Entity References: An entity reference contains a name between the start and the end delimiters. For example **&amp;** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

Character References: These contain references, such as **&#65;**, contains a hash mark (“#”) followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

XML Text

- The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
- To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
- Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.

- Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below:

not allowed character	**replacement-entity**	**character description**
<	&lt;	less than
>	&gt;	greater than
&	&amp;	Ampersand
'	&apos;	Apostrophe
"	&quot;	quotation mark

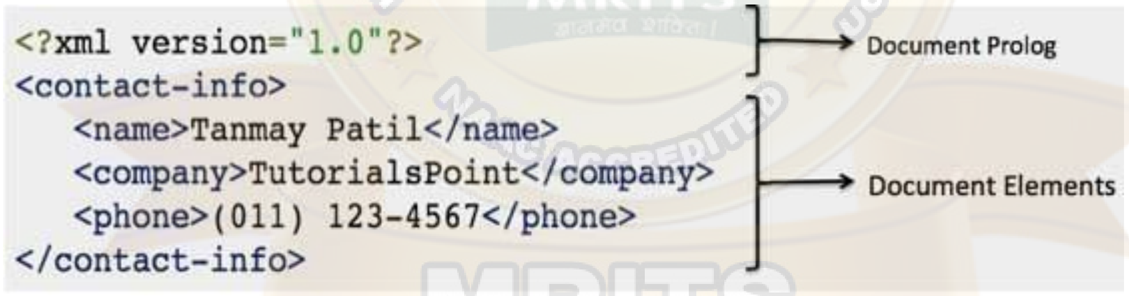
An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document example

A simple document is given in the following example:

```
<?xml version="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

The following image depicts the parts of XML document.



```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Document Prolog Section

The **document prolog** comes at the top of the document, before the root element. This section contains:

- XML declaration
- Document type declaration

You can learn more about XML declaration here : XML Declaration.

Document Elements Section

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

This chapter covers XML *declaration* in detail. XML declaration contains details that prepare an XML processor to parse the XML document. It is optional, but when used, it must appear in first line of the XML document.

Syntax

Following syntax shows XML declaration:

```
<?xml
  version="version_number"
  encoding="encoding_declaration"
  standalone="standalone_status"
?>
```

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote. Following table shows the above syntax in detail:

Parameter	Parameter_value	Parameter_description
Version	1.0	Specifies the version of the XML standard used.
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	*yes* or *no*.	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to *no*. Setting it to *yes* tells the processor there are no external declarations required for parsing the document.

Rules

An XML declaration should abide with the following rules:

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: *version, encoding and standalone.*
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

XML Declaration Examples

Following are few examples of XML declarations:

XML declaration with no parameters:

<?xml >

XML declaration with version definition:

<?xml version="1.0">

XML declaration with all parameters defined:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

XML declaration with all parameters defined in single quotes:

<?xml version='1.0' encoding='iso-8859-1' standalone='no'?>

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. An example of start-tag is:

<address>

End Tag

Every element that has a start tag should end with an end-tag. An example of end-tag is:

</address>

Note that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called *content*. An element which has no content is termed as **empty**. An **empty** element can be represented in two ways as below:

(1) A start-tag immediately followed by an end-tag as shown below:

<hr></hr>

(2) A complete empty-element tag is as shown below:

<hr/>

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags:

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and the end tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example:

```
<outer_element>
<internal_element>
    This tag is closed before the outer_element
</internal_element>
</outer_element>
```

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.

Syntax

Following is the syntax to write an XML element:

```
<element-nameattribute1attribute2>
....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:

```
<nameattribute1attribute2.../>
```

Example of an XML document using various XML element:

```
<?xml version="1.0"?>
<contact-info>
<addresscategory="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.

Syntax

An XML attribute has 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, 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*. Hence we have two plant elements with different attributes.

You can also observe that we have declared this attribute at the beginning of the 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 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 either less than sign <

Comments can be used to include related links, information and terms. They are visible only in the source code; not in the XML code. Comments may appear anywhere in XML code.

Syntax

XML comment has following syntax:

```
<!-------Your comment---- >
```

A comment starts with <!-- and ends with -->. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

Example

Following example demonstrates the use of comments in XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!---Students grades are uploaded by months --- >
<class_list>
<student>
<name>Tanmay</name>
<grade>A</grade>
</student>
</class_list>
```

Any text between <!-- and --> characters is considered as a comment.

XML Comments Rules

Following rules are needed to be followed for XML comments:

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments must not appear within attribute values.
- Comments cannot be nested inside the other comments.

The document entity serves as the root of the entity tree and a starting-point for an XML processor.

This means, entities are the placeholders in XML. These can be declared in the document prolog or in a DTD. There are different types of entities and this chapter will discuss Character Entity.

Both, the HTML and the XML, have some symbols reserved for their use, which cannot be used as content in XML code. For example, < and > signs are used for opening and closing XML tags. To display these special characters, the character entities are used.

There are few special characters or symbols which are not available to be typed directly from keyboard. Character Entities can be used to display those symbols/special characters also.

Types of Character Entities

There are three types of character entities:

- Predefined Character Entities
- Numbered Character Entities
- Named Character Entities

Predefined Character Entities

They are introduced to avoid the ambiguity while using some symbols. For example, an ambiguity is observed when less than (<) or greater than (>) symbol is used with the angle tag(<>). Character entities are basically used to delimit tags in XML. Following is a list of pre-defined character entities from XML specification. These can be used to express characters without ambiguity.

- Ampersand: &amp;
- Single quote: &apos;
- Greater than: &gt;
- Less than: &lt;
- Double quote: &quot;

Numeric Character Entities

The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format. As there are thousands of numeric references available, these are a bit hard to remember. Numeric reference refers to the character by its number in the Unicode character set.

General syntax for decimal numeric reference is:

&#decimal number ;

General syntax for hexadecimal numeric reference is:

&#x Hexadecimal number ;

The following table lists some predefined character entities with their numeric values:

Entity name	Character	Decimal reference	Hexadecimal reference
Quot	"	&#34;	&#x22;
Amp	&	&#38;	&#x26;
Apos	'	&#39;	&#x27;
Lt	<	&#60;	&#x3C;
Gt	>	&#62;	&#x3E;

Named Character Entity

As its hard to remember the numeric characters, the most preferred type of character entity is the named character entity. Here, each entity is identified with a name.

For example:

- 'Aacute' represents capital Á character with acute accent.
- 'ugrave' represents the small ù with grave accent.

The predefined entities such as &lt;, &gt;, and &amp; require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Syntax

Following is the syntax for CDATA section:

```
<![CDATA[
  characters with markup
]]>
```

The above syntax is composed of three sections:

- **CDATA Start section -** CDATA begins with the nine-character delimiter **<![CDATA[**
- **CDATA End section -** CDATA section ends with **]]>** delimiter.
- **CData section -** Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

```
<script>
<![CDATA[
<message>Welcome to TutorialsPoint</message>
]]>
</script>
```

In the above syntax, everything between <message> and </message> is treated as character data and not as markup.

CDATA Rules

The given rules are required to be followed for XML CDATA:

- CDATA cannot contain the string "]]>" anywhere in the XML document.

- Nesting is not allowed in CDATA section.

XML document contain two types of white spaces **(a)** Significant Whitespace and **(b)** Insignificant Whitespace. Both are explained below with examples.

Significant Whitespace

A significant Whitespace occurs within the element which contain text and markup present together. For example:

<name>TanmayPatil</name>

and

<name>Tanmay Patil</name>

The above two elements are different because of the space between **Tanmay** and **Patil**. Any program reading this element in an XML file is obliged to maintain the distinction.

Insignificant Whitespace

Insignificant whitespace means the space where only element content is allowed. For example:

<address.category="residence">

or

<address... category="..residence">

The above two examples are same. Here, the space is represented by dots (**.**). In the above example, the space between *address* and *category* is insignificant.

A special attribute named **xml:space** may be attached to an element. This indicates that whitespace should not be removed for that element by the application. You can set this attribute to **default** or **preserve** as shown in the example below:

<!ATTLIST address xml:space (default|preserve) 'preserve'>

Where:

- The value **default** signals that the default whitespace processing modes of an application are acceptable for this element;
- The value **preserve** indicates the application to preserve all the whitespaces.

"Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application. Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Syntax

Following is the syntax of PI:

<?target instructions?>

Where:

- **target** - identifies the application to which the instruction is directed.
- **instruction** - it is a character that describes the information for the application to process.

A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string ?> is encountered.

Example

PIs are rarely used. They are mostly used to link XML document to a style sheet. Following is an example:
<?xml-stylesheet href="tutorialspointstyle.css" type="text/css"?>
Here, the *target* is xml-stylesheet. *href="tutorialspointstyle.css"* and *type="text/css"* are *data* or *instructions* that the target application will use at the time of processing the given XML document.
In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown; the first attribute states that the type of the transform is XSL and the second attribute points to its location.

Processing Instructions Rules

A PI can contain any data except the combination ?>, which is interpreted as the closing delimiter. Here are two examples of valid PIs:
<?welcome to pg=10 of tutorials point?>

<?welcome?>
"Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application. Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Syntax

Following is the syntax of PI:
<?target instructions?>
Where:

- **target** - identifies the application to which the instruction is directed.
- **instruction** - it is a character that describes the information for the application to process.

A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string ?> is encountered.

Example

PIs are rarely used. They are mostly used to link XML document to a style sheet. Following is an example:
<?xml-stylesheet href="tutorialspointstyle.css" type="text/css"?>
Here, the *target* is xml-stylesheet. *href="tutorialspointstyle.css"* and *type="text/css"* are *data* or *instructions* that the target application will use at the time of processing the given XML document.
In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown; the first attribute states that the type of the transform is XSL and the second attribute points to its location.

Processing Instructions Rules

A PI can contain any data except the combination ?>, which is interpreted as the closing delimiter. Here are two examples of valid PIs:

```
<?welcome to pg=10 of tutorials point?>

<?welcome?>
```

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are:

- Well-formed XML document
- Valid XML document

Well-formed XML document

An XML document is said to be **well-formed** if it adheres to the following rules:

- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>... </title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Example

Example of well-formed XML document:

```
<?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>
```

Above example is said to be well-formed as:

- It defines the type of document. Here, the document type is **element** type.
- It includes a root element named as **address**.

- Each of the child elements among name, company and phone is enclosed in its self explanatory tag.
- Order of the tags is maintained.

Valid XML document – Document Type Definition

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. he 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 external source.

Syntax

The syntax of internal DTD is as shown:

```
<!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 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** are 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 are written as below:

<!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 Schema

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:elementname="contact">
<xs:complexType>
<xs:sequence>
<xs:elementname="name"type="xs:string"/>
<xs:elementname="company"type="xs:string"/>
<xs:elementname="phone"type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:elementname="x"type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:elementname="phone_number"type="xs:int"/>
```

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:elementname="Address">
<xs:complexType>
<xs:sequence>
<xs:elementname="name"type="xs:string"/>
        <xs:elementname="company"type="xs:string"/>
<xs:elementname="phone"type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:elementname="AddressType">
<xs:complexType>
<xs:sequence>
<xs:elementname="name"type="xs:string"/>
        <xs:elementname="company"type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:elementname="Address1">
<xs:complexType>
<xs:sequence>
<xs:elementname="address"type="AddressType"/>
        <xs:elementname="phone1"type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:elementname="Address2">
<xs:complexType>
<xs:sequence>
<xs:elementname="address"type="AddressType"/>
        <xs:elementname="phone2"type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:

```
<xs:attribute
```

An XML document is always descriptive. The tree structure is often referred to as XML Tree and plays an important role to describe any XML document easily.

The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root.

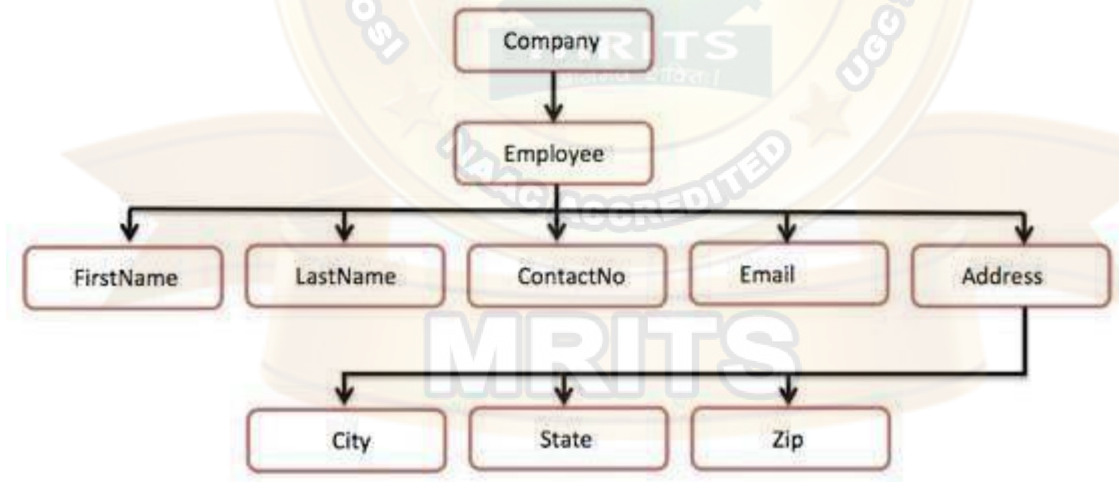
The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes.

Example

Following example demonstrates simple XML tree structure:

```
<?xml version="1.0"?>
<Company>
<Employee>
<FirstName>Tanmay</FirstName>
<LastName>Patil</LastName>
<ContactNo>1234567890</ContactNo>
<Email>tanmaypatil@xyz.com</Email>
<Address>
<City>Bangalore</City>
<State>Karnataka</State>
<Zip>560212</Zip>
</Address>
</Employee>
</Company>
```

Following tree structure represents the above XML document:



In the above diagram, there is a root element named as <company>. Inside that, there is one more element <Employee>. Inside the employee element, there are five branches named <FirstName>, <LastName>, <ContactNo>, <Email>, and <Address>. Inside the <Address> element, there are three sub-branches, named <City><State> and <Zip>.

The Document Object Model

The Document Object Model (DOM) is the foundation of XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a way of describing those nodes and the relationships between them.

A DOM Document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be *tree based*.
The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

Example

The following example (sample.htm) parses an XML document ("address.xml") into an XML DOM object and then extracts some information from it with JavaScript:

```
<!DOCTYPE html>
<html>
<body>
<h1>TutorialsPoint DOM example </h1>
<div>
<b>Name:</b><spanid="name"></span><br>
<b>Company:</b><spanid="company"></span><br>
<b>Phone:</b><spanid="phone"></span>
</div>
<script>
if(window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp =newXMLHttpRequest();
}
else
{// code for IE6, IE5
        xmlhttp =newActiveXObject("Microsoft.XMLHTTP");
}
      xmlhttp.open("GET","/xml/address.xml",false);
      xmlhttp.send();
      xmlDoc=xmlhttp.responseXML;

      document.getElementById("name").innerHTML=
      xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
      document.getElementById("company").innerHTML=
      xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
      document.getElementById("phone").innerHTML=
      xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

Contents of **address.xml** are as below:

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

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

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

Namespace Declaration

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

```
<elementxmlns: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:contactxmlns: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 Database is used to store the huge amount of information in the XML format. As the use of XML is increasing in every field, it is required to have the secured place to store the XML documents. The data stored in the database can be queried using **XQuery**, serialized, and exported into desired format.

XML Database Types

There are two major types of XML databases:

- XML- enabled
- Native XML (NXD)

XML- Enabled Database

XML enabled database is nothing but the extension provided for the conversion of XML document. This is relational database, where data are stored in tables consisting of rows and columns. The tables contain set of records, which in turn consist of fields.

Native XML Database

Native XML database is based on the container rather than table format. It can store large amount of XML document and data. Native XML database is queried by the **XPath**-expressions.

Native XML database has advantage over the XML-enabled database. It is highly capable to store, query and maintain the XML document than XML-enabled database.

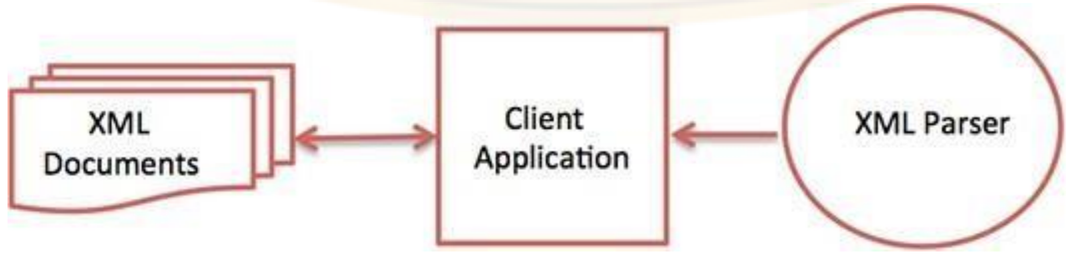
Example

Following example demonstrates XML database:

```
<?xml version="1.0"?>
<contact-info>
<contact1>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact1>
<contact2>
<name>Manisha Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 789-4567</phone>
</contact2>
</contact-info>
```

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document:



The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below:

- **MSXML (Microsoft Core XML Services) :** This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument :** This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser :** The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon :** Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces :** Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

XHTML

XHTML stands for **EXtensible HyperText Markup Language.** It is a cross between HTML and XML language.

XHTML is almost identical to HTML but it is stricter than HTML. XHTML is HTML defined as an XML application. It is supported by all major browsers.

Although XHTML is almost the same as HTML but It is more important to create your code correctly, because XHTML is stricter than HTML in syntax and case sensitivity. XHTML documents are well-formed and parsed using standard XML parsers, unlike HTML, which requires a lenient HTML-specific parser.

XHTML was developed to make HTML more extensible and increase interoperability with other data formats. There are two main reasons behind the creation of XHTML:

- It creates a stricter standard for making web pages, reducing incompatibilities between browsers. So it is compatible for all major browsers.
- It creates a standard that can be used on a variety of different devices without changes.

Let's take an example to understand it.

HTML is mainly used to create web pages but we can see that many pages on the internet contain "bad" HTML (not follow the HTML rule).

This HTML code works fine in most browsers (even if it does not follow the HTML rules).

For example:

```
<html>
<head>
<title>This is an example of bad HTML</title>
<body>
<h1>Bad HTML
<p>This is a paragraph
</body>
```

The above HTML code doesn't follow the HTML rule although it runs. Now a day, there are different browser technologies. Some browsers run on computers, and some browsers run on mobile phones or other small devices. The main issue with the bad HTML is that it can't be interpreted by smaller devices.

So, XHTML is introduced to combine the strengths of HTML and XML.

XHTML is HTML redesigned as XML. It helps you to create better formatted code on your site. XHTML doesn't facilitate you to make badly formed code to be XHTML compatible. Unlike with HTML (where simple errors (like missing out a closing tag) are ignored by the browser), XHTML code must be exactly how it is specified to be.

XHTML Syntax

XHTML syntax is very similar to HTML syntax and all the valid HTML elements are also valid in XHTML. But XHTML is case sensitive so you have to pay a bit extra attention while writing an XHTML document to make your HTML document compliant to XHTML.

You must remember the following important points while writing a new XHTML document or converting existing HTML document into XHTML document:

- All documents must have a DOCTYPE.
- All tags must be in lower case.
- All documents must be properly formed.
- All tags must be closed.
- All attributes must be added properly.
- The name attribute has changed.
- Attributes cannot be shortened.
- All tags must be properly nested.

DOCTYPE Declaration

All XHTML documents must contain a DOCTYPE declaration at the start. There are three types of DOCTYPE declarations:

- Strict
- Transitional
- Frameset

Here is an example of using DOCTYPE.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

Tags must be in lower case

XHTML is case-sensitive markup language. So, all the XHTML tags and attributes must be written in lower case.

<!-- Invalid in XHTML -->

<A Href="/xhtml/xhtml_tutorial.html">XHTML Tutorial**</A>**

<!-- Valid in XHTML -->

<a href="/xhtml/xhtml_tutorial.html">XHTML Tutorial**</a>**

Closing Tags are mandatory

An XHTML must have an equivalent closing tag. Even empty elements should also have closing tags. Let's see an example:

<!-- Invalid in XHTML -->

<p>This paragraph is not written according to XHTML syntax.

```
<!-- Invalid in XHTML -->
<img src="/images/xhtml.gif" >
<!-- Valid in XHTML -->
<p>This paragraph is not written according to XHTML syntax.</p>
<!-- Valid in XHTML-->
<img src="/images/xhtml.gif" />
```

Attribute Quotes

All the XHTML attribute's values must be quoted. Otherwise, your XHTML document is assumed as an invalid document.

See this example:

```
<!-- Invalid in XHTML -->
<img src="/images/xhtml.gif" width=250 height=50 />
<!-- Valid in XHTML -->
<img src="/images/xhtml.gif" width="250" height="50" />
```

Attribute Minimization

XHTML doesn't allow you to minimize attributes. You have to explicitly state the attribute and its value.

See this example:

```
<!--Invalid in XHTML -->
<option selected>
<!-- valid in XHTML-->
<option selected="selected">
```

A list of minimized attributes in HTML and the way you need to write them in XHTML.

HTML Style	XHTML Style
compact	compact="compact"
checked	checked="checked"
declare	declare="declare"
readonly	readonly="readonly"
disabled	disabled="disabled"
selected	selected="selected"
defer	defer="defer"

ismap	ismap="ismap"
nohref	nohref="nohref"
noshade	noshade="noshade"
nowrap	nowrap="nowrap"
multiple	multiple="multiple"
noresize	noresize="noresize"

The id Attribute
The id attribute is used to replace the name attribute. Instead of using name = "name", XHTML prefers to use id = "id".
See this example:
<!-- Invalid in XHTML -->
<img src="/images/xhtml.gif" name="xhtml_logo" />
<!-- Valid in XHTML -->
<img src="/images/xhtml.gif" id="xhtml_logo" />

The language attribute
In XHTML, the language attribute of script tag is deprecated so you have to use type attribute instead of this.
See this example:
<!-- Invalid in XHTML -->
<script language="JavaScript" type="text/JavaScript">
document.write("Hello XHTML!");
</script>
<!-- Valid in XHTML -->
<script type="text/JavaScript">
document.write("Hello XHTML!");
</script>

Nested Tags
XHTML tags must be nested properly. Otherwise your document is assumed as an incorrect XHTML document.
See this example:
<!-- Invalid in XHTML -->
<b><i> This text is bold and italic**</b></i>**
<!-- Valid in XHTML -->

<b><i> This text is bold and italic**</i></b>**

Element Prohibitions

The following elements are not allowed to have any other element inside them. This is applicable for all the descending elements.

Element	**Prohibition**
<a>	It must not contain other <a> elements.
<pre>	It must not contain the <img>, <object>, <big>, <small>, <sub>, or <sup> elements.
<button>	It must not contain the <input>, <select>, <textarea>, <label>, <button>, <form>, <fieldset>, <iframe> or <isindex> elements.
<label>	It must not contain other <label> elements.
<form>	It must not contain other <form> elements.

UNIT III

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

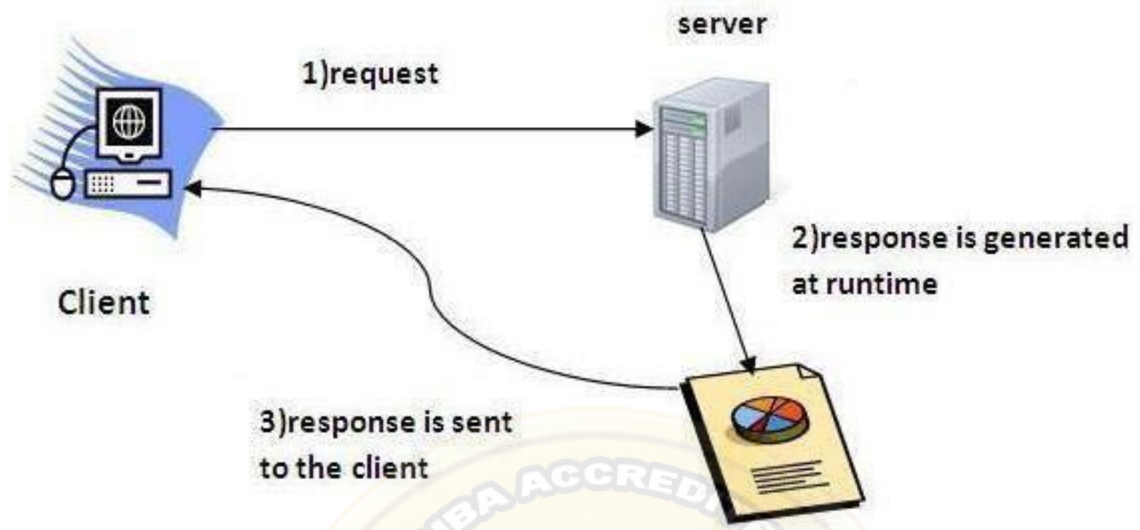
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

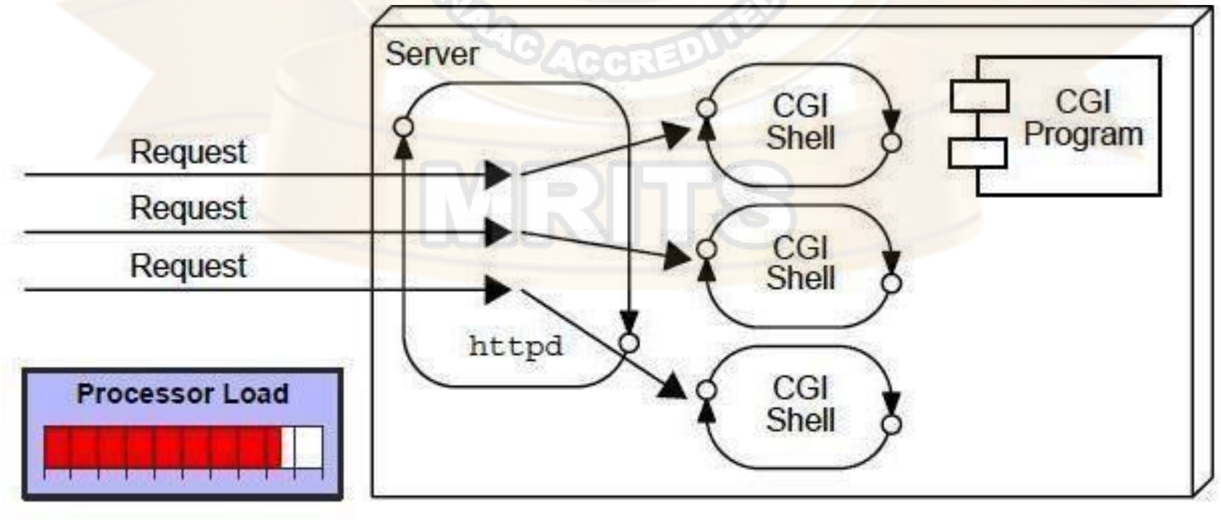


What is web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI(Commmon Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

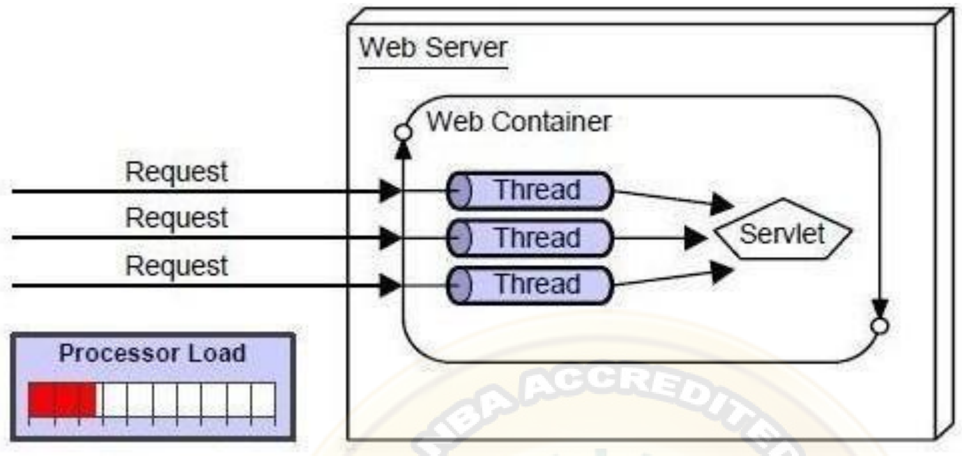


Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet



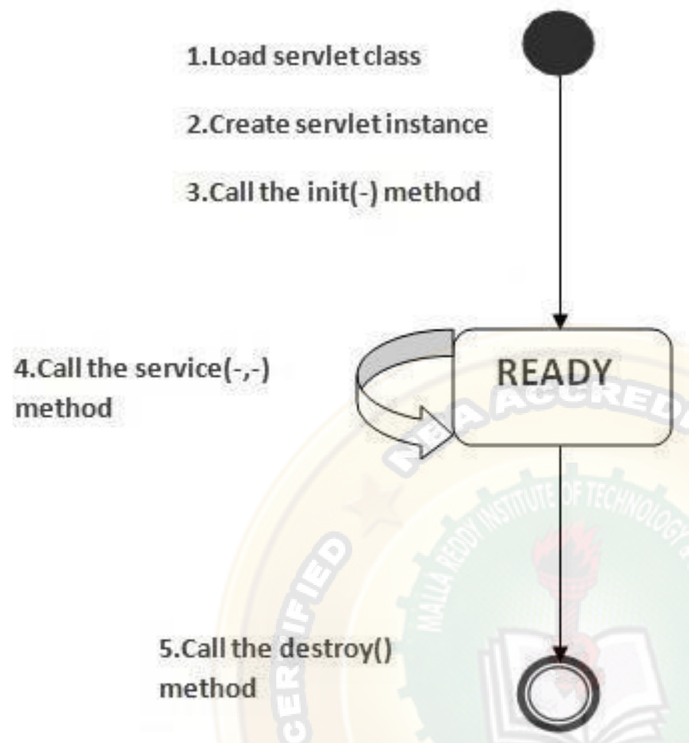
There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language..

Life Cycle of a Servlet

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded
The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
2) Servlet instance is created
The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) **throws** ServletException
4) service method is invoked
The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service

method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:
public void service(ServletRequest request, ServletResponse response) **throws** ServletExcepti on, IOException
5) destroy method is invoked
The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:
public void destroy()

Deploying a Servlet

Steps to create a servlet example

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.
The servlet example can be created by three ways:

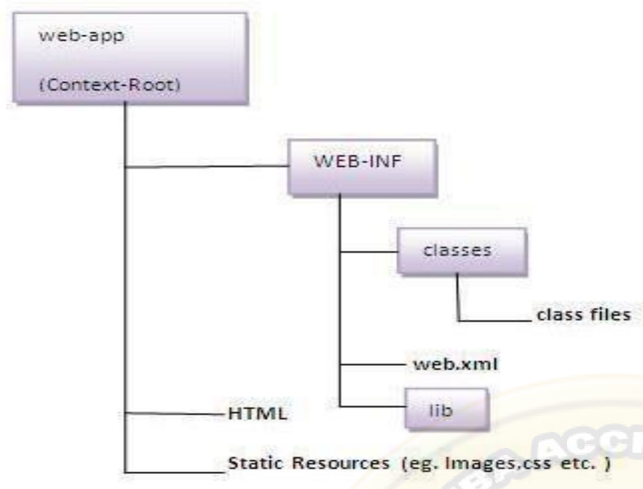
1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1) Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.
The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2) Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close();//closing the stream
}}
```

3) Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	**Server**
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4) Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
```

```
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5) Start the Server and deploy the project

Copy the project and paste it in the webapps folder under apache tomcat.

But there are several ways to deploy the project. They are as follows:

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

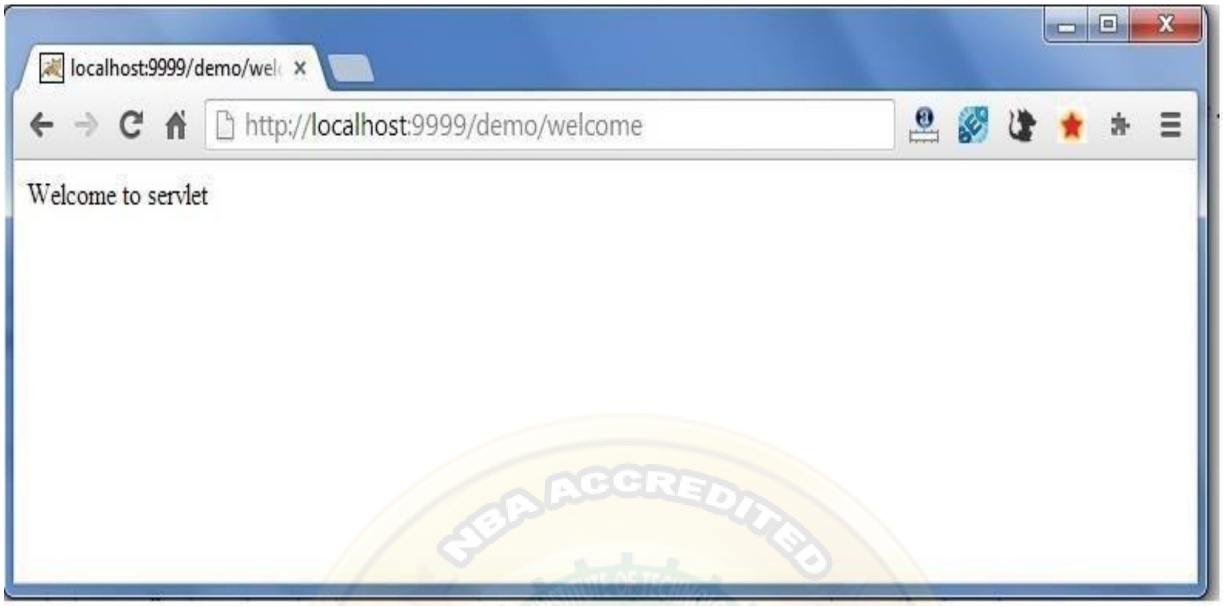
You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

```
projectfolder> jar cvf myproject.war *
```

Creating war file has an advantage that moving the project from one location to another takes less time.

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:
http://localhost:9999/demo/welcome



Welcome to servlet

Servlet API

1. Servlet API
2. Interfaces in javax.servlet package
3. Classes in javax.servlet package
4. Interfaces in javax.servlet.http package
5. Classes in javax.servlet.http package

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener

12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1 GenericServlet
2 ServletInputStream
3 ServletOutputStream
4 ServletRequestWrapper
5 ServletResponseWrapper
6 ServletRequestEvent
7 ServletContextEvent
8 ServletRequestAttributeEvent
9 ServletContextAttributeEvent
10 ServletException
11 UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1 HttpServletRequest
2 HttpServletResponse
3 HttpSession
4 HttpSessionListener
5 HttpSessionAttributeListener
6 HttpSessionBindingListener
7 HttpSessionActivationListener
8 HttpSessionContext (deprecated now)
9 Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1 HttpServlet
2 Cookie
3 HttpServletRequestWrapper
4 HttpServletResponseWrapper
5 HttpSessionEvent
6 HttpSessionBindingEvent

7 HttpUtils (deprecated now)

Servlet Interface

Servlet interface provides common behaviour to all the servlets.
Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet Example by implementing Servlet interface

Let's see the simple example of servlet by implementing the servlet interface.
First.java

```
import java.io.*;
import javax.servlet.*;

public class First implements Servlet{
ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("servlet is initialized");
}

public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
```

```
res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");

}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}

}
```

Web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>First</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>

</web-app>
```

GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable**interfaces. It provides the implementation of all the methods of these interfaces except the service method.
GenericServlet class can handle any type of request so it is protocol-independent.
You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.
Methods of GenericServlet class
There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.

2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

Let's see the simple example of servlet by inheriting the GenericServlet class.

First.java

```
import java.io.*;
import javax.servlet.*;

public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");

}
}
```

Web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>First</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>

</web-app>
```

HttpServlet class

1. HttpServlet class
2. Methods of HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.

9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Reading Servlet parameters

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GETMethod

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?**(question mark) symbol as follows −

```
http://www.test.com/hello?key1 = value1&key2 = value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

POSTMethod

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

Reading Form Datausing Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

GETMethodExampleusingURL

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI
Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
publicclassHelloFormextendsHttpServlet{

publicvoid doGet(HttpServletRequest request,HttpServletResponse response)
throwsServletException,IOException{

// Set response content type
    response.setContentType("text/html");

PrintWriterout= response.getWriter();
String title ="Using GET Method to Read Form Data";
String docType =
"<!doctype html public \"-//w3c//dtd html 4.0 "+"transitional//en\">\n";

out.println(docType +
"<html>\n"+
"<head><title>"+ title +"</title></head>\n"+
"<body bgcolor = \"#f0f0f0\">\n"+
"<h1 align = \"center\">"+ title +"</h1>\n"+
"<ul>\n"+
"  <li><b>First Name</b>: "
+ request.getParameter("first_name")+"\n"+
" <li><b>Last Name</b>: "
+ request.getParameter("last_name")+"\n"+
"</ul>\n"+
"</body>
</html>"
);
}
}
```

Assuming your environment is set up properly, compile HelloForm.java as follows −

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class**file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloForm</servlet-name>
<servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloForm</servlet-name>
<url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type *http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI*in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result −

Using GETMethod to Read Form Data

- **First Name**: ZARA
- **Last Name**: ALI

GETMethodExampleUsingForm

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this imput.

```
<html>
<body>
<formaction="HelloForm"method="GET">
        First Name: <inputtype="text"name="first_name">
<br/>
        Last Name: <inputtype="text"name="last_name"/>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

First Name: Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POSTMethodExampleUsingForm

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
publicclassHelloFormextendsHttpServlet{
// Method to handle GET method request.
publicvoid doGet(HttpServletRequest request,HttpServletResponse response)
throwsServletException,IOException{
// Set response content type
    response.setContentType("text/html");
PrintWriterout= response.getWriter();
String title ="Using GET Method to Read Form Data";
String docType =
"<!doctype html public \"-//w3c//dtd html 4.0 "+
"transitional//en\">\n";
out.println(docType +
"<html>\n"+
"<head><title>"+ title +"</title></head>\n"+
"<body bgcolor = \"#f0f0f0\">\n"+
"<h1 align = \"center\">"+ title +"</h1>\n"+
"<ul>\n"+
"  <li><b>First Name</b>: "
+ request.getParameter("first_name")+"\n"+
" <li><b>Last Name</b>: "
+ request.getParameter("last_name")+"\n"+
"</ul>\n"+
"</body>
</html>"
);
}

// Method to handle POST method request.
publicvoid doPost(HttpServletRequest request,HttpServletResponse response)
throwsServletException,IOException{

    doGet(request, response);
}
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

```
<html>
<body>
<formaction="HelloForm"method="POST">
      First Name: <inputtype="text"name="first_name">
<br/>
      Last Name: <inputtype="text"name="last_name"/>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name: [ ] Last Name: [ ]

Based on the input provided, it would generate similar result as mentioned in the above examples.

Reading Initialization parameters

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Syntax of getServletConfig() method

```
public ServletConfig getServletConfig();
```

Example of getServletConfig() method

```
ServletConfig config=getServletConfig();
//Now we can call the methods of ServletConfig interface
```

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>
<servlet>
......

<init-param>
<param-name>parametername</param-name>
<param-value>parametervalue</param-value>
</init-param>
......
</servlet>
</web-app>
```

Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

response.setContentType("text/html");
PrintWriter out = response.getWriter();

ServletConfig config=getServletConfig();
String driver=config.getInitParameter("driver");
out.print("Driver is: "+driver);
out.close();
}

}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>

<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

</web-app>
```

Handling HTTP Request & Responses

DETAILS ABOUT REQUEST & RESPONSE INTERFACES.

HttpServlet gets request from a client (browser) on the HttpServletRequest interface and can return response to the client through the HttpServletResponse interface in all the doXXXX methods.

HERE IS SOME OF MOST USED HTTPSERVLETREQUEST METHODS:

Method	**Description**
public Enumeration getParameterNames()	Names of the parameters contained in this request.
public String[] getParameterValues(String name)	Used when the named parameter may have multiple values.
public Map getParameterMap()	Returns all the parameters stored in a Map

	object.
public BufferedReader getReader()	This will retrieve the body of a request as characters into a BufferedReader.
public ServletInputStream getInputStream()	This will retrieve the body of a request as binary data into a ServletInputStream.
public Enumeration getHeaderNames()	Returns an enumeration of all the header names this request contains.
public String getQueryString()	This will return the query string that is contained in the request URL after the path.
public RequestDispatcher getRequestDispatcher(String path)	A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response.

You have to use either the getReader() method or the getInputStream(). You cannot combine to use these for the same request.

HERE IS SOME OF MOST USED HTTPSERVLETRESPONSE METHODS:

Method	**Description**
public PrintWriter getWriter()	This will return a PrintWriter object that can send character text to the client.
public ServletOutputStream getOutputStream()	This will return a ServletOutputStream suitable for writing binary data in the response.
public addHeader(String name, String value)	This will add a property, which is a String name with a String value to the response header.
public addDateHeader(String name, long date)	This will add a property, which is a String name with a long date value to the response header.
public void sendRedirect(String location)	This will send a temporary redirect response to the client using the specified redirect location URL.
public Enumeration getHeaderNames()	Returns an enumeration of all the header names this request contains.

You have to use either the getWriter() method or the getOutputStream(). You cannot combine to use these for the same response

EXAMPLE OF SERVLET HANDLING A HTML FORM INPUT.

AS SELECTED WE USE NETBEANS IDE AND GLASSFISH SERVER.

You can download this example here *(needed tools can be found in the right menu on this page)*.

If you like to participate in the review of this example you must first create a Web project in Netbeans(the project name is ServletFormhandling).

Simple servlet handling a html form input example:

```
package app.form;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass FormHandlerServlet extends HttpServlet
{
  @Override
protectedvoid doPost(HttpServletRequest request, HttpServletResponse response)
  {
    String enteredValue;
// gets all the selected options from the client browser
    String[] selectedOptions = request.getParameterValues("options");
// gets the enteredValue fields value
    enteredValue = request.getParameter("enteredValue");
    response.setContentType("text/html");
    PrintWriter printWriter;
try
    {
// get a printwriter from the HttpServletResponse objects ref.
      printWriter = response.getWriter();
// return on the HttpServletResponse objects ref. requested values
      printWriter.println("<p>");
      printWriter.print("You entered: ");
      printWriter.print(enteredValue);
      printWriter.print("</p>");
      printWriter.println("<p>");
      printWriter.print("The following options were selected:");
      printWriter.println("<br/>");

if (selectedOptions != null)
```

```
      {
for (String option : selectedOptions)
        {
          printWriter.print(option);
          printWriter.println("<br/>");
        }
      }
else
      {
        printWriter.println("None");
      }
      printWriter.println("</p>");
    }
catch (IOException e)
    {
      e.printStackTrace();
    }
  }
            }
```

For those who participate in the review: create a Servlet in Netbeans and replace generated code for the servlet with that shown above (the servlet name is FormHandlerServlet).

- In this servlet we expect a POST Http request and we fetch all the values we know about from a html form (read the comments).
- We set the Content type to "text/html" for the response to the client (browser) and use a PrintWriter on the response object to return html codes and text back to the client.

We need a html startup file, **dataentry.html**, for the browser like this:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="windows-1252">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Data Entry Page</title>
</head>
<body>
<form method="post" action="FormHandlerServlet"
          style="width: 310px; border: 1px solid black;">
<table >
<tr>
<td>Please enter some text:</td>
<td><input type="text" name="enteredValue" />
```

```
</td>
</tr>
<tr>
<td colspan="2">
<input name="options" type="checkbox" value="option1" />
        Option 1</td>
</tr>
<tr>
<td colspan="2">
<input name="options" type="checkbox" value="option2" />
        Option 2</td>
</tr>
<tr>
<td colspan="2">
<input name="options" type="checkbox" value="option3" />
        Option 3</td>
</tr>
<tr>
<td colspan="2" style="text-align: right;">
<input type="submit" value="Submit">
</td>
</tr>
</table>
</form>
</body>
</html>
```

For those who participate in the review: create a HTML page in Netbeans and replace generated code for the html file with that shown above (the name of the html should be dataentry and places in the folder web).

CREATING DEPLOYMENT DESCRIPTOR.

- To run this Servlet you have to deploy it to a web-server or a Application server. To deploy means to install the Servlet with some instruction to a such server.
- The instructions are mainly defined to be deployment descriptors. The standard part of the deployment descriptor should be in an XML-file with the name web.xml. In the later version of java it is possible to specify this in the form of annotations in front of the Servlet.
- The contents of the web.xml file regarding servlet, FormHandlerServlet, should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
<servlet>
<servlet-name>FormHandlerServlet</servlet-name>
<servlet-class>app.form.FormHandlerServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FormHandlerServlet</servlet-name>
<url-pattern>/FormHandlerServlet</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>
     30
</session-timeout>
</session-config>
<welcome-file-list>
<welcome-file>dataentry.html</welcome-file>
</welcome-file-list>
     </web-app>
```

- This file starts with the normal xml tag for a XML file and the root tag for the deployment descriptor is web-app. Every ting inside the last tag is to tell the server about our application, which in this case is a Servlet.
- With a servlet tag we give the Servlet class a servlet name, which is used in the servlet-mapping tag to specify a url for the Servlet.
- In this way we can have many urls for the same servlet.
- If no session-timeout (the server ends the service of the application after this time) is given a standard timeout for the server is used as timeout for the application.
- The welcome-file tag specifies the startup for our application, which in this case and our application is the welcome file dataentry.html. **Reorganize the welcome-file-list** to what is shown above.

CREATING WEB-SERVER DEPLOYMENT DESCRIPTOR.

- The context-root (in example /ServletFormhandling) for the application will in most cases be specified by a server vendor deployment descriptor.

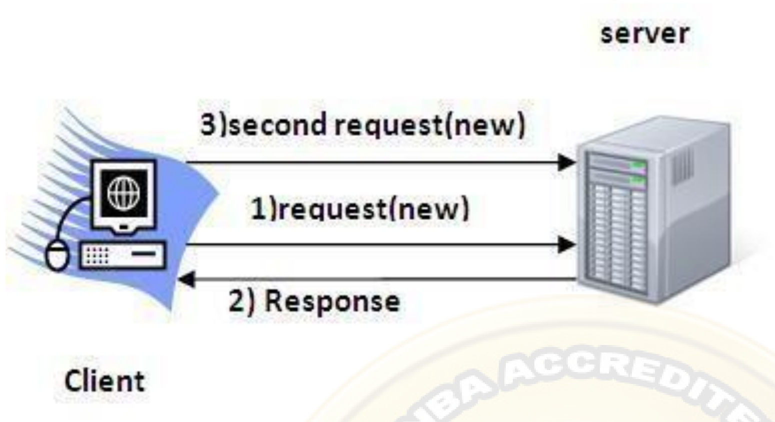
Using Cookies and Sessions

Session simply means a particular interval of time.

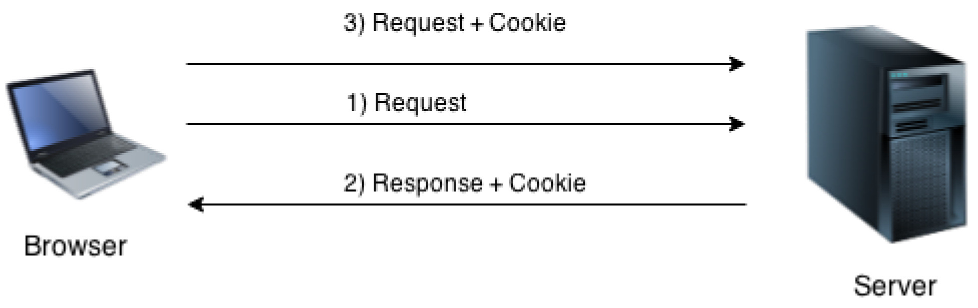
Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?
To recognize the user It is used to recognize the particular user.
Session Tracking Techniques
There are four techniques used in Session tracking:
Cookies
Hidden Form Field
URL Rewriting
HttpSession
Cookies in Servlet
A cookie is a small piece of information that is persisted between the multiple client requests.
A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
How Cookie works
By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie
There are 2 types of cookies in servlets.

Non-persistent cookie
Persistent cookie
Non-persistent cookie
It is valid for single session only. It is removed each time when user closes the browser.
Persistent cookie
It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.
Advantage of Cookies
Simplest technique of maintaining the state.
Cookies are maintained at client side.
Disadvantage of Cookies
It will not work if cookie is disabled from the browser.
Only textual information can be set in Cookie object.
Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.
Cookie class
javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class
There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void	changes the value of the cookie.

setValue(String value)	

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.

public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

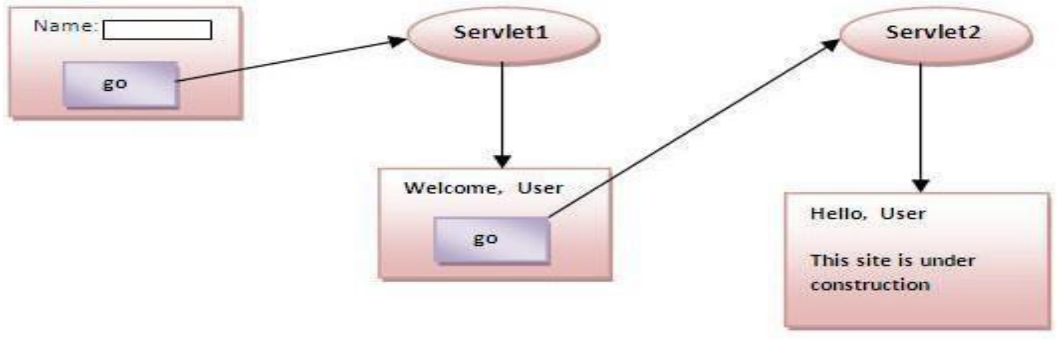
How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cooki
e
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
try{

response.setContentType("text/html");
PrintWriter out = response.getWriter();

String n=request.getParameter("userName");
out.print("Welcome "+n);

Cookie ck=new Cookie("uname",n);//creating cookie object
response.addCookie(ck);//adding cookie in the response

//creating submit button
out.print("<form action='servlet2'>");
out.print("<input type='submit' value='go'>");
out.print("</form>");

out.close();
}catch(Exception e){System.out.println(e);}
}
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
try{

response.setContentType("text/html");
PrintWriter out = response.getWriter();

Cookie ck[]=request.getCookies();
out.print("Hello "+ck[0].getValue());

out.close();

}catch(Exception e){System.out.println(e);}
}


}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
```

```
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

Session

- HttpSession Interface provides a way to identify a user across more than one pagerequest or visit to a Web site and to store information about that user.
- Web container creates a session id for each user. The container uses this id to identifythe particular user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.

Get the HttpSession object

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **publicHttpSessiongetSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **publicHttpSessiongetSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Destroy Session

session.**invalidate**();

Set/Get data in session

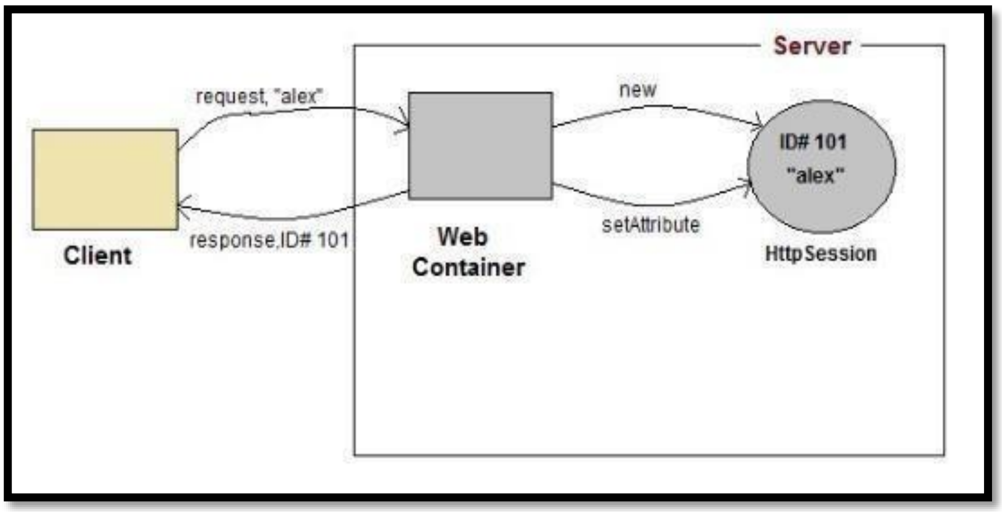
session**.setAttribute(name,value);**
session**.getAttribute(name);**

Methods

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Steps

- On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by webcontainer.
- The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
- The **Web Container** uses this ID, finds the matching session with the IDand associates the session with therequest.



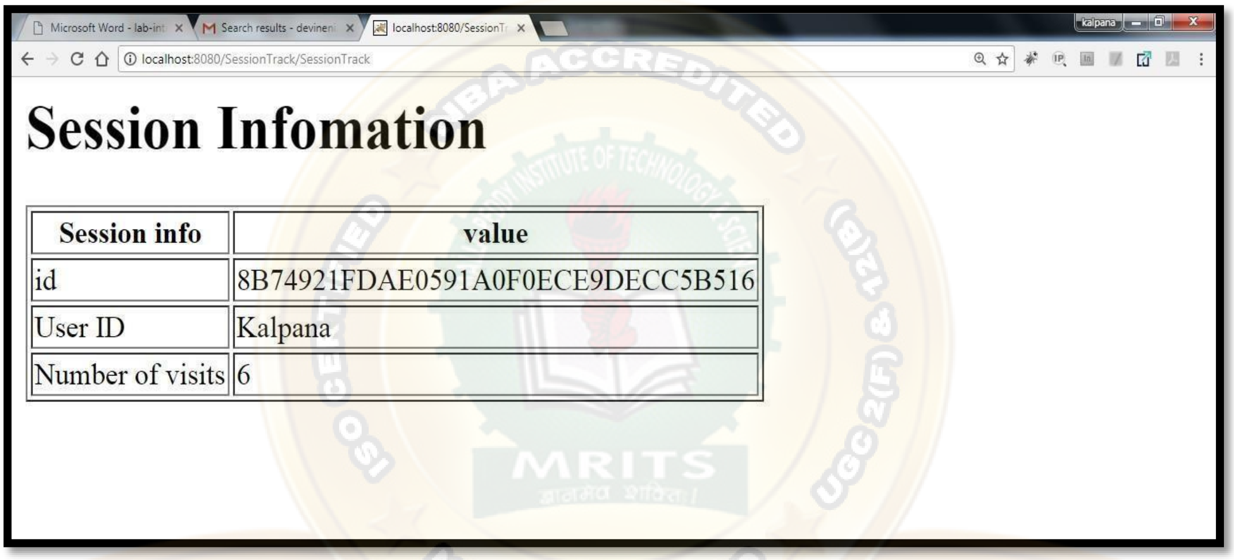
Ex: SessionTrack.java

```
import java.io.*;
importjavax.servlet.*;
importjavax.servlet.http.*;

public class SessionTrack extends HttpServlet {

public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    // Create a session object if it is already not created.
HttpSession session = request.getSession(true);

    String title = "Welcome to my website";
String userID = "";
    Integer visitCount = new Integer(0);

if (session.isNew())
{
userID = "Kalpana";
session.setAttribute("UserId", "Kalpana");
    }
 else {
visitCount = (Integer)session.getAttribute("visitCount");
visitCount = visitCount + 1;
userID = (String)session.getAttribute("UserId");
    }
session.setAttribute("visitCount", visitCount);

response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>" +
           "<body>" +
           "<h1>Session Infomation</h1>" +
           "<table border='1'>" +
           "<tr><th>Session info</th><th>value</th></tr>" +
           "<tr><td>id</td><td>" + session.getId() + "</td></tr>" +
           "<tr><td>User ID</td<td>" + userID + –</td></tr>" +
           "<tr><td>Number of visits</td><td>" + visitCount + "</td></tr>" +
           "</table></body></html>");
  }
}
```

we b.xml

```
<web-app>
          <servlet>
          <servlet-name>SessionTrack</servlet-name>
          <servlet-class>SessionTrack</servlet-class>
          </servlet>
          <servlet-mapping>
          <servlet-name>SessionTrack</servlet-name>
          <url-pattern>/SessionTrack</url-pattern>
          </servlet- mapping>
</web-app>
```

Output:



Session Infomation

Session info	value
id	8B74921FDAE0591A0F0ECE9DECC5B516
User ID	Kalpana
Number of visits	6

Connecting to database using JDBC

To start with basic concept, let us create a simple table and create few records in that table as follows −

CreateTable

To create the **Employees** table in TEST database, use the following steps −

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

```
C:\>
C:\>cd ProgramFiles\MySQL\bin
C:\Program Files\MySQL\bin>
```

Step 2

Login to database as follows

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password:********
mysql>
```

Step 3

Create the table **Employee** in **TEST** database as follows −

```
mysql>use TEST;
mysql> create table Employees(
  id intnotnull,
  age intnotnull,
  first varchar (255),
last varchar (255)
);
Query OK,0 rows affected (0.08 sec)
mysql>
```

CreateDataRecords

Finally you create few records in Employee table as follows −

```
mysql> INSERT INTO Employees VALUES (100,18,'Zara','Ali');
Query OK,1 row affected (0.05 sec)
mysql> INSERT INTO Employees VALUES (101,25,'Mahnaz','Fatma');
Query OK,1 row affected (0.00 sec)
mysql> INSERT INTO Employees VALUES (102,30,'Zaid','Khan');
Query OK,1 row affected (0.00 sec)
mysql> INSERT INTO Employees VALUES (103,28,'Sumit','Mittal');
Query OK,1 row affected (0.00 sec)
mysql>
```

AccessingaDatabase

Here is an example which shows how to access TEST database using Servlet.

```
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
publicclassDatabaseAccessextendsHttpServlet{
publicvoid doGet(HttpServletRequest request,HttpServletResponse response)
throwsServletException,IOException{
// JDBC driver name and database URL
staticfinalString JDBC_DRIVER ="com.mysql.jdbc.Driver";
staticfinalString DB_URL="jdbc:mysql://localhost/TEST";
// Database credentials
staticfinalString USER ="root";
staticfinalString PASS ="password";
// Set response content type
    response.setContentType("text/html");
PrintWriterout= response.getWriter();
String title ="Database Result";
String docType =
"<!doctype html public \"-//w3c//dtd html 4.0 "+"transitional//en\">\n";
out.println(docType +
"<html>\n"+
"<head><title>"+ title +"</title></head>\n"+
"<body bgcolor = \"#f0f0f0\">\n"+
"<h1 align = \"center\">"+ title +"</h1>\n");
try{
// Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
```

```
// Open a connection
Connection conn =DriverManager.getConnection(DB_URL, USER, PASS);
// Execute SQL query
Statement stmt = conn.createStatement();
String sql;
        sql ="SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
// Extract data from result set
while(rs.next()){
//Retrieve by column name
int id = rs.getInt("id");
int age = rs.getInt("age");
String first =rs.getString("first");
Stringlast= rs.getString("last");
//Display values
out.println("ID: "+ id+"<br>");
out.println(", Age: "+ age +"<br>");
out.println(", First: "+ first +"<br>");
out.println(", Last: "+last+"<br>");
}
out.println("</body></html>");
// Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
}catch(SQLException se){
//Handle errors for JDBC
        se.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forName
        e.printStackTrace();
}finally{
//finally block used to close resources
try{
if(stmt!=null)
             stmt.close();
}catch(SQLException se2){
}// nothing we can do
try{
if(conn!=null)
          conn.close();
}catch(SQLException se){
        se.printStackTrace();
}//end finally try
}//end try
}
}
```

Now let us compile above servlet and create following entries in web.xml

```
....
<servlet>
<servlet-name>DatabaseAccess</servlet-name>
<servlet-class>DatabaseAccess</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DatabaseAccess</servlet-name>
<url-pattern>/DatabaseAccess</url-pattern>
</servlet-mapping>
....
```

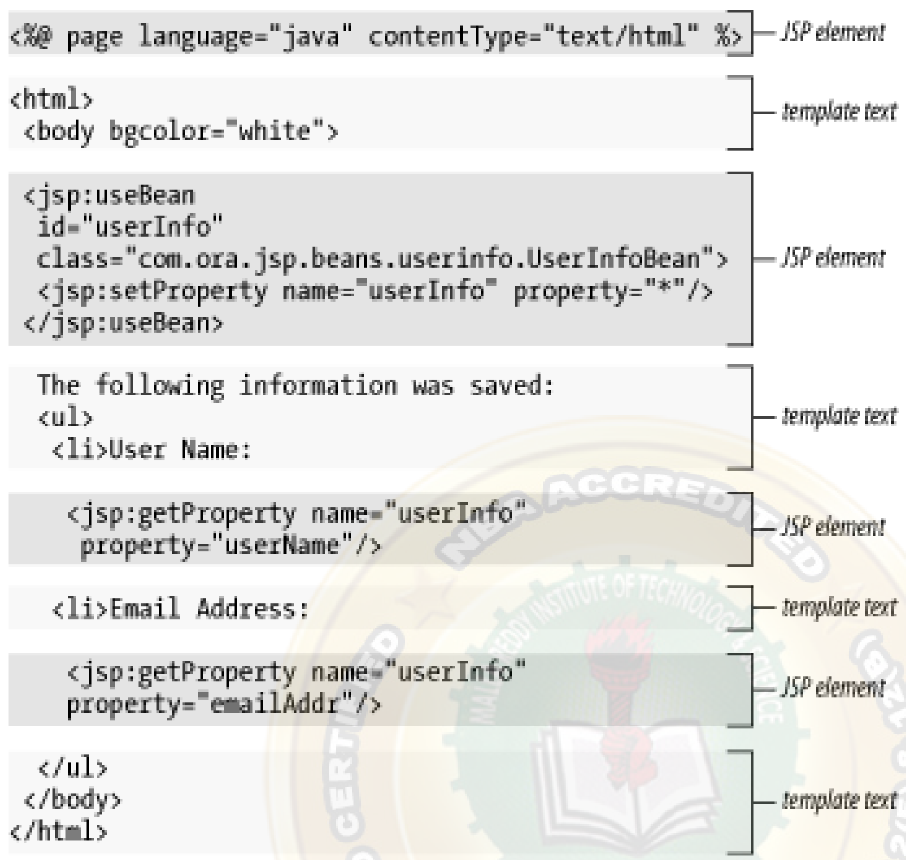
Now call this servlet using URL http://localhost:8080/DatabaseAccess which would display following response −

```
DatabaseResult

ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

UNIT IV

The Anatomy of a JSP Page

A JSP page is simply a regular web page with JSP elements for generating the parts that differ for each request,



```
<%@ page language="java" contentType="text/html" %>

<html>
 <body bgcolor="white">

 <jsp:useBean
  id="userInfo"
  class="com.ora.jsp.beans.userinfo.UserInfoBean">
  <jsp:setProperty name="userInfo" property="*"/>
 </jsp:useBean>

  The following information was saved:
  <ul>
   <li>User Name:

    <jsp:getProperty name="userInfo"
     property="userName"/>

   <li>Email Address:

    <jsp:getProperty name="userInfo"
    property="emailAddr"/>

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

Everything in the page that isn't a JSP element is called *template text*. Template text can be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web-page language in use today, most of the descriptions and examples in this book use HTML, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.

When a JSP page request is processed, the template text and dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

JSP Processing

Just as a web server needs a servlet container to provide an interface to servlets, the server needs a *JSP container* to process JSP pages.

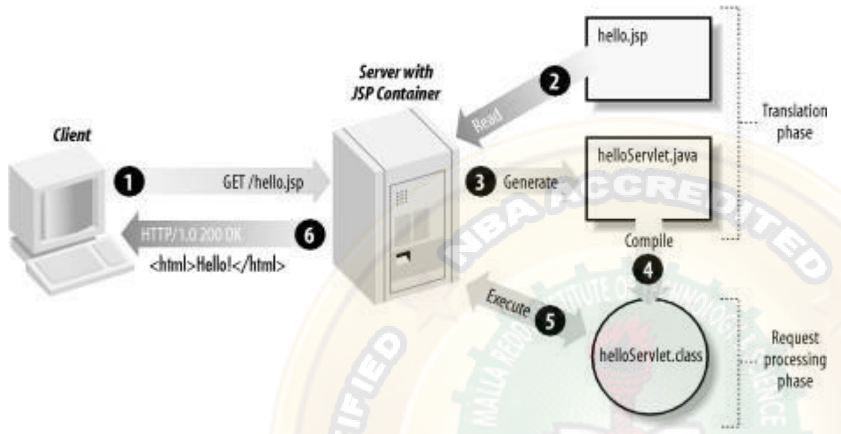
The JSP container is responsible for intercepting requests for JSP pages. To process all JSP elements in the page, the container first turns the JSP page into a servlet (known as the *JSP page implementation class*).

The conversion is pretty straightforward; all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior. The container then compiles the servlet class.

Converting the JSP page to a servlet and compiling the servlet form the *translation phase*.

The JSP container initiates the translation phase for a page automatically when it receives the first request for the page. Since the translation phase takes a bit of time, the first user to request a JSP page notices a slight delay.

The translation phase can also be initiated explicitly; this is referred to as *precompilation* of a JSP page. Precompiling a JSP page is a way to avoid hitting the first user with this delay.
The JSP container is also responsible for invoking the JSP page implementation class (the generated servlet) to process each request and generate the response. This is called the *requestprocessing phase*.
JSP page translation and processing phases

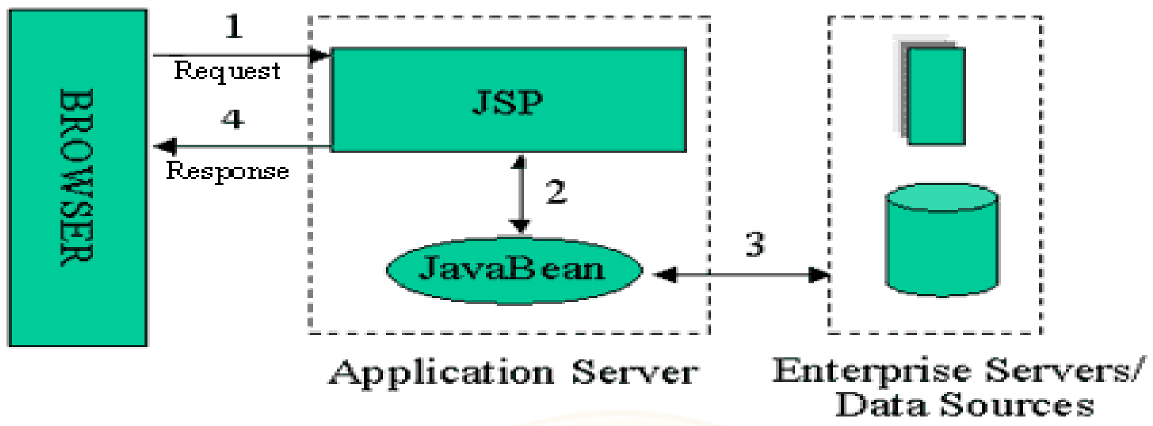


As long as the JSP page remains unchanged, any subsequent request goes straight to the request processing phase (i.e., the container simply executes the class file). When the JSP page is modified, it goes through the translation phase again before entering the request processing phase.
The JSP container is often implemented as a servlet configured to handle all requests for JSP pages. In fact, these two containers--a servlet container and a JSP container--are often combined in one package under the name *web container*. a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet: it's loaded once and called repeatedly, until the server is shut down. By virtue of being an automatically generated servlet, a JSP page inherits all the advantages of a servlet: platform and vendor independence, integration, efficiency, scalability, robustness, and security.

JSP Application Design with MVC

. The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller.
In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing.
Business logic is the term used for the manipulation of an application's data, such as customer, product, and order information.
Presentation refers to how the application data is displayed to the user, for example, position, font, and size.
And finally, *request processing* is what ties the business logic and presentation parts together.

JSP Model 1 Architecture



In MVC terms, the Model corresponds to business logic and data, the View to the presentation, and the Controller to the request processing.

Why use this design with JSP? The answer lies primarily in the first two elements. Remember that an application data structure and logic (the Model) is typically the most stable part of an application, while the presentation of that data (the View) changes fairly often. Just look at all the face-lifts many web sites go through to keep up with the latest fashion in web design. Yet, the data they present remains the same.

Another common example of why presentation should be separated from the business logic is that you may want to present the data in different languages or present different subsets of the data to internal and external users.

Access to the data through new types of devices, such as cell phones and personal digital assistants (PDAs), is the latest trend. Each client type requires its own presentation format. It should come as no surprise, then, that separating business logic from the presentation makes it easier to evolve an application as the requirements change; new presentation interfaces can be developed without touching the business logic.

JSDK

1 Set classpath to <home dir>/jsdk2.0/lib/jsdk.jar;
2. Set path to <home dir>/jsdk2.0/bin;
3. Compile the servlet program (.java file).
4. Copy the .class file to <home dir>/jsdk2.0/examples
5. Execute servletrunner
6. Open web browser
7. Enter the url as follows in the address bar: http://host:8080/servlet/<servlet name>

In this unit we focus on how to develop applications using JSP. JSP syntax is a fluid mix of two basic content forms: *scriptlet elements* and *markup*. Markup is typically standard HTML or XML, while scriptlet elements are delimited blocks of Java code which may be intermixed with the markup. When the page is requested the Java code is executed and its output is added, in situ, with the surrounding markup to create the final page. JSP pages must be compiled to Java bytecode classes before they can be executed, but such compilation is needed only when a change to the source JSP file has occurred.

Contents

JSP Elements

There are three types of JSP elements you can use: *scripting,action* and *directive*.

Scripting elements

Scripting elements allow you to add small pieces of code (typically Java code) in a JSP page, such as an if statement to generate different HTML depending on a certain condition.

Like actions, they are also executed when the page is requested. You should use scripting elements with extreme care: if you embed too much code in your JSP pages, you will end up with the same kind of maintenance problems as with servlets embedding HTML.

Scripting elements

Element	Description
<% ... %>	Scriptlet, used to embed scripting code.
<%= ... %>	Expression, used to embed scripting code expressions when the result shall be added to the response. Also used as request-time action attribute values.
<%! ... %>	Declaration, used to declare instance variables and methods in the JSP page implementation class.

Standard action elements

Action elements typically perform some action based on information that is required at the exact time the JSP page is requested by a browser. An action can, for instance, access parameters sent with the request to do a database lookup. It can also dynamically generate HTML, such as a table filled with information retrieved from an external system.

The JSP specification defines a few standard action elements

Action element	Description
<jsp:useBean>	Makes a JavaBeans component available in a page
<jsp:getProperty>	Gets a property value from a JavaBeans component and adds it to the response
<jsp:setProperty>	Sets a JavaBeans component property value
<jsp:include>	Includes the response from a servlet or JSP page during the request processing phase
<jsp:forward>	Forwards the processing of a request to servlet or JSP page
<jsp:param>	Adds a parameter value to a request handed off to another servlet or JSP page using <jsp:include> or <jsp:forward>
<jsp:plugin>	Generates HTML that contains the appropriate browser-dependent elements

	(OBJECT or EMBED) needed to execute an applet with the Java Plug-in software

Custom action elements and the JSP Standard Tag Library

In addition to the standard actions, the JSP specification includes a Java API a programmer can use to develop *custom actions* to extend the JSP language. The JSP Standard Tag Library (JSTL) is such an extension, with the special status of being defined by a formal specification from Sun and typically bundled with the JSP container. JSTL contains action elements for processes needed in most JSP applications, such as conditional processing, database access, internationalization, and more.

If JSTL isn't enough, programmers on your team (or a third party) can use the extension API to develop additional custom actions, may be to access application-specific resources or simplify application-specific processing.

Directive elements

The directive elements specify information about the page itself that remains the same between requests--for example, if session tracking is required or not, buffering requirements, and the name of a page that should be used to report errors, if any.

Directive elements

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

JSP Implicit objects

JSP object	**Servlet API Object**	**Description**
application	javax.servlet.ServletContext	Context (Execution environment) of the Servlet.
config	javax.servlet.ServletConfig	The ServletConfig for the JSP.
exception	java.lang.Throwable	The exception that resulted when an error occurred.
out	javax.servlet.jsp.JspWriter	An object that writes into a JSP's output stream.
pageContext	javax.servlet.jsp.PageContext	Page context for the JSP.

request	javax.servlet.HttpServletRequest	The client request.
response	javax.servlet.HttpServletResponse	The response to the client.
session	javax.servlet.http.HttpSession	Session object created for requesting client.
page	javax.servlet.Servlet	Refers to current servlet object.

Using beans in JSP pages

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods.

```
package mypack;
```

```
public class Test{
public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());

}}
```

Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

1. <jsp:useBean id= "instanceName" scope= "page | request | session | application"
2. **class**= "packageName.className" type= "packageName.className"
3. beanName="packageName.className | <%= expression >" >
4. </jsp:useBean>

Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
   - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
   - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
   - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
   - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

Simple example of jsp:useBean action tag

In this example, we are simply invoking the method of the Bean class.

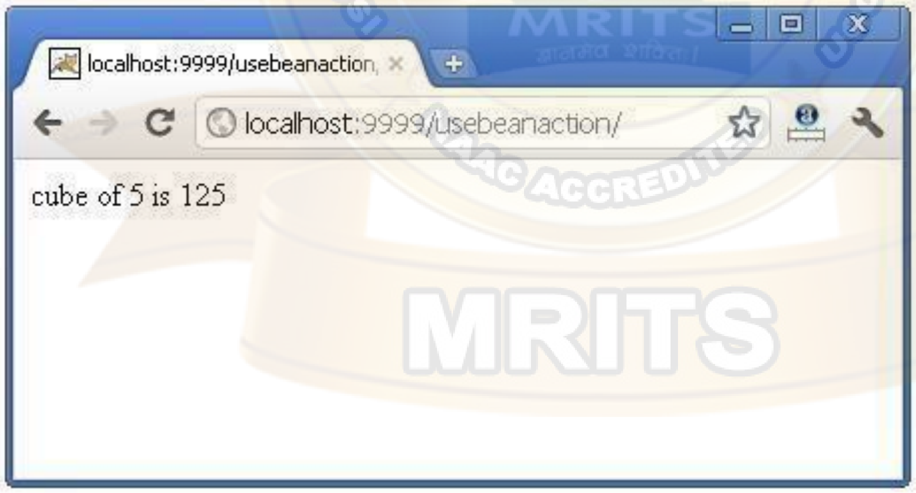
For the example of setProperty, getProperty and useBean tags, visit next page.

Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{

public int cube(int n){return n*n*n;}

}
```

index.jsp file

```
    <jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

    <%
    int m=obj.cube(5);
    out.print("cube of 5 is "+m);
%>
```



```
cube of 5 is 125
```

jsp:setProperty and jsp:getProperty action tags

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.

The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

<jsp:setProperty name="instanceOfBean" property= "*" |

```
property="propertyName" param="parameterName" |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean

```
<jsp:setProperty name="bean" property="*" />
```

Example of jsp:setProperty action tag if you have to set value of the incoming specific property

```
<jsp:setProperty name="bean" property="username" />
```

Example of jsp:setProperty action tag if you have to set a specific value in the property

```
<jsp:setProperty name="bean" property="username" value="Kumar" />
```

jsp:getProperty action tag

The jsp:getProperty action tag returns the value of the property.

Syntax of jsp:getProperty action tag

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Simple example of jsp:getProperty action tag

```
<jsp:getProperty name="obj" property="name" />
```

Example of bean development in JSP

In this example there are 3 pages:

- index.html for input of values
- welocme.jsp file that sets the incoming values to the bean object and prints the one value
- User.java bean class that have setter and getter methods

index.html

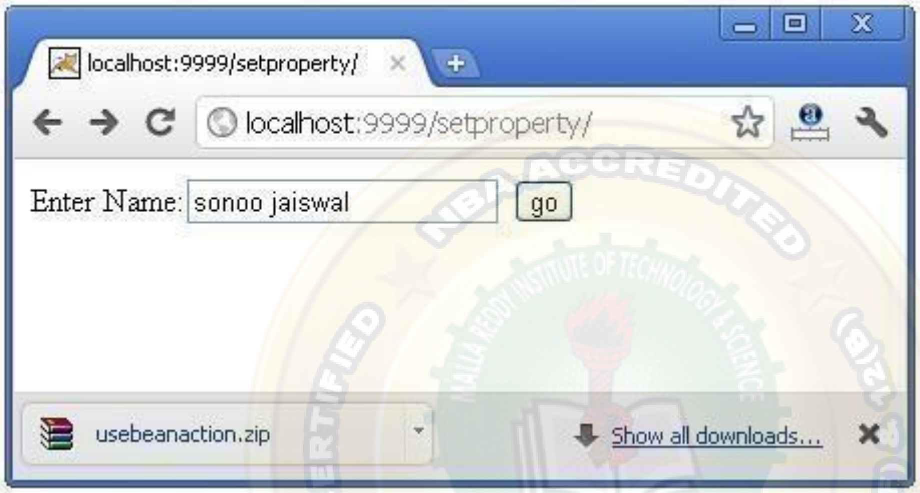
```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

process.jsp

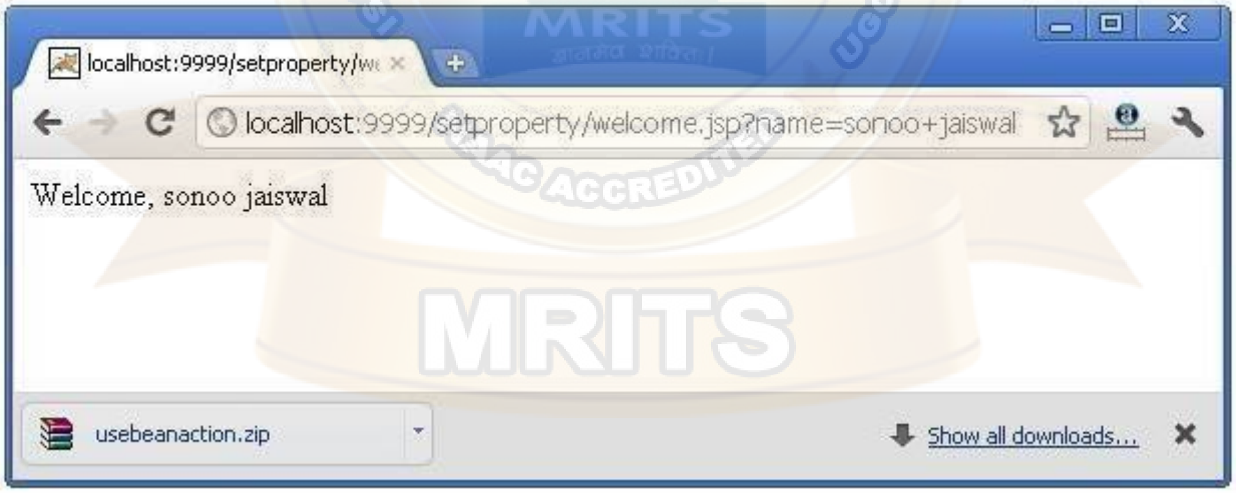
```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

User.java

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```



Enter Name: sonoo jaiswal go



localhost:9999/setproperty/w
localhost:9999/setproperty/welcome.jsp?name=sonoo+jaiswal

Welcome, sonoo jaiswal

usebeanaction.zip
Show all downloads...

Cookies in JSP

Cookies are text files stored on the client computer and they are kept for various information tracking purposes. JSP transparently supports HTTP cookies using underlying servlet technology.

There are three steps involved in identifying and returning users –

- Server script sends a set of cookies to the browser. For example, name, age, or identification number, etc.
- Browser stores this information on the local machine for future use.
- When the next time the browser sends any request to the web server then it sends those cookies information to the server and server uses that information to identify the user or may be for some other purpose as well.

This chapter will teach you how to set or reset cookies, how to access them and how to delete them using JSP programs.

TheAnatomyofaCookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A JSP that sets a cookie might send headers that look something like this −

```
HTTP/1.1200 OK
Date:Fri,04Feb200021:03:38 GMT
Server:Apache/1.3.9(UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires =Friday,04-Feb-0722:03:38 GMT;
   path =/; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the **Set-Cookie header** contains **a name value pair, a GMT date, a path** and **a domain**. The name and value will be URL encoded. The **expires** field is an instruction to the browser to **"forget"** the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection:Keep-Alive
User-Agent:Mozilla/4.6(X11; I;Linux2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126

Accept: image/gif,*/*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name = xyz
```

A JSP script will then have access to the cookies through the request method ***request.getCookies()*** which returns an array of *Cookie* objects.

Cookies Methods

Following table lists out the useful methods associated with the Cookie object which you can use while manipulating cookies in JSP −

S.No.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which the cookie applies; for example, tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which the cookie applies; for example, tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	**public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, **-1** indicating the cookie will persist until the browser shutdown.
5	**public String getName()** This method returns the name of the cookie. The name cannot be changed after the creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie.
7	**public String getValue()** This method gets the value associated with the cookie.
8	**public void setPath(String uri)** This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	**public String getPath()** This method gets the path to which this cookie applies.
10	**public void setSecure(boolean flag)** This method sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e, SSL) connections.
11	**public void setComment(String purpose)** This method specifies a comment that describes a cookie's purpose. The

	comment is useful if the browser presents the cookie to the user.
12	**public String getComment()** This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

SettingCookieswithJSP

Setting cookies with JSP involves three steps −

Step 1: Creating a Cookie object

You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Keep in mind, neither the name nor the value should contain white space or any of the following characters −

```
[ ] ( ) = , " / ? @ : ;
```

Step 2: Setting the maximum age

You use **setMaxAge** to specify how long (in seconds) the cookie should be valid. The following code will set up a cookie for 24 hours.

```
cookie.setMaxAge(60*60*24);
```

Step 3: Sending the Cookie into the HTTP response headers

You use **response.addCookie** to add cookies in the HTTP response header as follows

```
response.addCookie(cookie);
```

Example

Let us modify our Form Example to set the cookies for the first and the last name.

```
<%
// Create cookies for first and last names.
Cookie firstName =newCookie("first_name", request.getParameter("first_name"));
Cookie lastName =newCookie("last_name", request.getParameter("last_name"));

// Set expiry date after 24 Hrs for both the cookies.
   firstName.setMaxAge(60*60*24);
   lastName.setMaxAge(60*60*24);

// Add both the cookies in the response header.
   response.addCookie( firstName );
   response.addCookie( lastName );
%>

<html>
```

```
<head>
<title>Setting Cookies</title>
</head>
<body>
<center>
<h1>Setting Cookies</h1>
</center>
<ul>
<li><p><b>First Name:</b>
<%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
<%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

Let us put the above code in **main.jsp** file and use it in the following HTML page −

```
<html>
<body>
<formaction="main.jsp"method="GET">
        First Name: <inputtype="text"name="first_name">
<br/>
        Last Name: <inputtype="text"name="last_name"/>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Keep the above HTML content in a file **hello.jsp** and put **hello.jsp** and **main.jsp** in **<Tomcat-installation-directory>/webapps/ROOT** directory. When you will access ***http://localhost:8080/hello.jsp***, here is the actual output of the above form.

First Name:

Last Name:

Try to enter the First Name and the Last Name and then click the submit button. This will display the first name and the last name on your screen and will also set two cookies **firstName** and **lastName**. These cookies will be passed back to the server when the next time you click the Submit button.

In the next section, we will explain how you can access these cookies back in your web application.

ReadingCookieswithJSP

To read cookies, you need to create an array of *javax.servlet.http.Cookie*objects by calling the **getCookies( )** method of *HttpServletRequest*. Then cycle through the array, and use **getName()** and **getValue()** methods to access each cookie and associated value.

Example

Let us now read cookies that were set in the previous example −

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
Cookie cookie =null;
Cookie[] cookies =null;
// Get an array of Cookies associated with the this domain
        cookies = request.getCookies();
if( cookies !=null){
out.println("<h2> Found Cookies Name and Value</h2>");
for(int i =0; i < cookies.length; i++){
              cookie = cookies[i];
out.print("Name : "+ cookie.getName()+",  ");
out.print("Value: "+ cookie.getValue()+" <br/>");
}
}else{
out.println("<h2>No cookies founds</h2>");
}
     %>
</body>

</html>
```

Let us now put the above code in **main.jsp** file and try to access it. If you set the **first_name cookie** as "John" and the **last_name cookie** as "Player" then running ***http://localhost:8080/main.jsp*** will display the following result −

```
FoundCookies NameandValue
```

```
Name : first_name, Value: John
Name : last_name, Value: Player
```

DeleteCookieswithJSP

To delete cookies is very simple. If you want to delete a cookie, then you simply need to follow these three steps −

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using the **setMaxAge()** method to delete an existing cookie.
- Add this cookie back into the response header.

Example

Following example will show you how to delete an existing cookie named **"first_name"** and when you run main.jsp JSP next time, it will return null value for first_name.

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
Cookie cookie =null;
Cookie[] cookies =null;
// Get an array of Cookies associated with the this domain
        cookies = request.getCookies();

if( cookies !=null){
out.println("<h2> Found Cookies Name and Value</h2>");

for(int i =0; i < cookies.length; i++){
            cookie = cookies[i];
if((cookie.getName()).compareTo("first_name")==0){
              cookie.setMaxAge(0);
              response.addCookie(cookie);
out.print("Deleted cookie: "+
              cookie.getName()+"<br/>");
}
out.print("Name : "+ cookie.getName()+", ");
out.print("Value: "+ cookie.getValue()+" <br/>");
}
}else{
out.println(
"<h2>No cookies founds</h2>");
}
```

```
    %>
</body>
</html>
```

Let us now put the above code in the **main.jsp** file and try to access it. It will display the following result −

```
CookiesNameandValue

Deleted cookie : first_name
Name : first_name, Value: John
Name : last_name, Value: Player
```

Now run *http://localhost:8080/main.jsp* once again and it should display only one cookie as follows −

```
FoundCookies NameandValue


Name : last_name, Value: Player
```

You can delete your cookies in the Internet Explorer manually. Start at the Tools menu and select the Internet Options. To delete all cookies, click the Delete Cookies button.

JSP- Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

MaintainingSessionBetweenWebClientAndServer

Let us now discuss a few options to maintain the session between the Web Client and the Web Server −

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

```
<input type = "hidden" name = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** or the **POST** data. Each time the web browser sends the request back, the **session_id** value can be used to keep the track of different web browsers.

This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting

You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.

For example, with **http://tutorialspoint.com/file.htm;sessionid=12345**, the session identifier is attached as **sessionid = 12345** which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

ThesessionObject

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

- a one page request or
- visit to a website or
- store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

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

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Here is a summary of important methods available through the session object −

S.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.

5	**public long getLastAccessedTime()** This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew()** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
<%@ page import="java.io.*,java.util.*" %>
<%
// Get session creation time.
Date createTime =newDate(session.getCreationTime());
// Get last access time of this Webpage.
Date lastAccessTime =newDate(session.getLastAccessedTime());
String title ="Welcome Back to my website";
Integer visitCount =newInteger(0);
String visitCountKey =newString("visitCount");
String userIDKey =newString("userID");
String userID =newString("ABCD");
```

```
// Check if this is new comer on your Webpage.
if(session.isNew()){
    title ="Welcome to my website";
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey, visitCount);
}
  visitCount =(Integer)session.getAttribute(visitCountKey);
  visitCount = visitCount +1;
  userID =(String)session.getAttribute(userIDKey);
  session.setAttribute(visitCountKey,  visitCount);
%>
<html>
<head>
<title>Session Tracking</title>
</head>
<body>
<center>
<h1>Session Tracking</h1>
</center>
<tableborder="1"align="center">
<trbgcolor="#949494">
<th>Session info</th>
<th>Value</th>
</tr>
<tr>
<td>id</td>
<td><%out.print( session.getId()); %></td>
</tr>
<tr>
<td>Creation Time</td>
<td><%out.print(createTime); %></td>
</tr>
<tr>
<td>Time of Last Access</td>
<td><%out.print(lastAccessTime); %></td>
</tr>
<tr>
<td>User ID</td>
<td><%out.print(userID); %></td>
</tr>
<tr>
```

```
<td>Number of visits</td>
<td><%out.print(visitCount); %></td>
</tr>
</table>
</body>
</html>
```

Now put the above code in **main.jsp** and try to access ***http://localhost:8080/main.jsp***. Once you run the URL, you will receive the following result −

Welcome to my website

Session Information

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Now try to run the same JSP for the second time, you will receive the following result.

Welcome Back to my website

Session Information

info type	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

DeletingSessionData

When you are done with a user's session data, you have several options −

- **Remove a particular attribute** − You can call the ***public void removeAttribute(String name)*** method to delete the value associated with the a particular key.
- **Delete the whole session** − You can call the ***public void invalidate()*** method to discard an entire session.

- **Setting Session timeout** − You can call the ***public void setMaxInactiveInterval(int interval)*** method to set the timeout for a session individually.
- **Log the user out** − The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration** − If you are using Tomcat, apart from the above mentioned methods, you can configure the session time out in web.xml file as follows.

```
<session-config>
<session-timeout>15</session-timeout>
</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.
The **getMaxInactiveInterval( )** method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, **getMaxInactiveInterval( )** returns 900.

Error Handling and Debugging
When you develop any application that's more than a trivial example, errors are inevitable. A JSP-based application is no exception. There are many types of errors you will deal with. Simple syntax errors in the JSP pages are almost a given during the development phase. And even after you have fixed all the syntax errors, you may still have to figure out why the application doesn't work as you intended because of design mistakes. The application must also be designed to deal with problems that can occur when it's deployed for production use. Users can enter invalid values and try to use the application in ways you never imagined. External systems, such as databases, can fail or become unavailable due to network problems.
Since a web application is the face of the company, making sure it behaves well, even when the users misbehave and the world around it falls apart, is extremely important for a positive customer perception. Proper design and testing is the only way to accomplish this goal.

connecting to databases in jsp

Database access in simple java programs, servlets ,and JSPs. We ca access the database by using JDBC. JDBC stands for "Java DataBase Connectivity". It is an API which consists of a set of Java classes, interfaces and exceptions and a specification to which both JDBC driver vendors and JDBC developers adhere when developing applications.

List of Drivers

- Bridge driver
    - sun.jdbc.odbc.JdbcOdbcDriver
        - jdbc:odbc:<dsn>
- Cloudscape
    - COM.cloudscape.core.JDBCDriver

    - jdbc:cloudscape:[database name and location]
- PostGRESQL
    - org.postgresql.Driver
        - jdbc:postgresql://[host]:[port]/[database name]
- MySQL
    - com.mysql.jdbc.Driver
        - jdbc:mysql://[host]:3306/[databasename]
- Oracle
    - oracle.jdbc.driver.OracleDriver

    jdbc:oracle:thin:@[host]:1521:[sid]

Steps to using Bridge driver

1. Create a data source name using ODBC
2. Load the database driver
3. Establish a Connection to the database
4. Create a Statement object
5. Execute SQL Query statement(s)
6. Retrieve the ResultSet Object
7. Retrieve record/field data from ResultSet
8. object for processing
9. Close ResultSet Object
10. Close Statement Object
11. Close Connection Object

javax.sql.RowSet

The interface that adds support to the JDBC API for the JavaBeans^{TM} component model. A rowset, which can be used as a JavaBeans component in a visual Bean development environment, can be created and configured at design time and executed at run time.

The RowSet interface provides a set of JavaBeans properties that allow a RowSet instance to be configured to connect to a JDBC data source and read some data from the data source. A group of setter methods (setInt, setBytes, setString, and so on) provide a way to pass input parameters to a rowset's command property. This command is the SQL query the rowset uses when it gets its data from a relational database, which is generally the case.

The RowSet interface supports JavaBeans events, allowing other components in an application to be notified when an event occurs on a rowset, such as a change in its value.

javax.sql.DataSource

A factory for connections to the physical data source that this DataSource object represents. An alternative to the DriverManager facility, a DataSource object is the preferred means of getting a connection. An object that implements the DataSource interface will typically be registered with a naming service based on the Java^{TM} Naming and Directory (JNDI) API.

The DataSource interface is implemented by a driver vendor. There are three types of implementations:

1. Basic implementation -- produces a standard Connection object
2. Connection pooling implementation -- produces a Connection object that will automatically participate in connection pooling. This implementation works with a middle-tier connection pooling manager.
3. Distributed transaction implementation -- produces a Connection object that may be used for distributed transactions and almost always participates in connection pooling. This implementation works with a middle-tier transaction manager and almost always with a connection pooling manager.

A DataSource object has properties that can be modified when necessary. For example, if the data source is moved to a different server, the property for the server can be changed. The benefit is that because the data source's properties can be changed, any code accessing that data source does not need to be changed.

A driver that is accessed via a DataSource object does not register itself with the DriverManager. Rather, a DataSource object is retrieved though a lookup operation and then used to create a Connection object. With a basic implementation, the connection obtained through a DataSource object is identical to a connection obtained through the DriverManager facility.

Specific database actions

```
<sql: transaction>

<sql: update>
   UPDATE Account SET Balance = Balance - 1000
     WHERE AccountNumber = 1234
</sql: update>
<sql: update>
   UPDATE Account SET Balance = Balance + 1000
     WHERE AccountNumber = 5678
</sql: update>

</sql: transaction>
```

All SQL actions that make up a transaction are placed in the body of a <sql:transaction> action element. This action tells the nested elements which database to use, so if you need to specify the database with the dataSource attribute, you must specify it for the <sql:transaction> action. The isolation attribute can specify special transaction features. When the Data Source is made available to the application through JNDI or by another application component, it's typically already configured with an appropriate isolation level. This attribute is therefore rarely used. The details of the different isolation levels are beyond the scope of this book. If you believe you need to specify this value, you can read up on the differences in the JDBC API documents or in the documentation for your database. You should also be aware that some databases and JDBC drivers don't support all transaction isolation levels.

Struts framework

Apache Struts is a free open-source framework for creating Java web applications. Web applications differ from conventional websites in that web applications can create a dynamic response. Many websites deliver only static pages. A web application can interact with databases and business logic engines to customize a response. Web applications based on JavaServer Pages sometimes commingle database code, page design code, and control flow code. In practice, we find that unless these concerns are separated, larger applications become difficult to maintain.

One way to separate concerns in a software application is to use a Model-View-Controller (MVC) architecture. The *Model* represents the business or database code, the *View* represents the page design code, and the *Controller* represents the navigational code. The Struts framework is designed to help developers create web applications that utilize a MVC architecture.

The framework provides three key components:

- A "request" handler provided by the application developer that is mapped to a standard URI.
- A "response" handler that transfers control to another resource which completes the response.
- A tag library that helps developers create interactive form-based applications with server pages.

The framework's architecture and tags are buzzword compliant. Struts works well with conventional REST applications and with nouveau technologies like SOAP and AJAX.

Configuring for Struts

1. Download the Struts binary release from http://jakarta.apache.org.
2. Extract the zip file.
3. Copy the .jar files to lib directory of the Web application.
4. Copy the .tld files to WEB-INF directory.
5. Store web.xml and struts-config.xml files in WEB-INF directory. (struts-config.xml is the DD for all Struts applications. It links all MVC components).

UNIT V

JavaScript – Introduction

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

Client-side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations ofJavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
JavaScript code
</script>
```

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 segment will look like −

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

Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
```

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

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons areOptional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<scriptlanguage="javascript"type="text/javascript">
<!--
    var1 =10
    var2 =20
//-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<scriptlanguage="javascript"type="text/javascript">
<!--
    var1 =10; var2 =20;
//-->
</script>
```

Note − It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE − Care should be taken while writing variable and function names in JavaScript.

Comments inJavaScript

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 <!--. JavaScript 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 //-->.

Example

The following example shows how to use comments in JavaScript.

```
<scriptlanguage="javascript"type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++
/*
    * This is a multiline comment in JavaScript
    * It is very similar to comments in C Programming
    */
//-->
</script>
```

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types −

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

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

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

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.

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

You can also declare multiple variables with the same **var** keyword as follows −

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

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.
For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

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

Note − Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.
JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>
<bodyonload= checkscope();>
<scripttype="text/javascript">
<!--
var myVar ="global";// Declare a global variable
function checkscope(){
var myVar ="local";// Declare a local variable
  document.write(myVar);
}
//-->
</script>
</body>
</html>
```

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

Abstract	else	instanceof	switch
Boolean	enum	int	synchronized
Break	export	interface	this
Byte	extends	long	throw
Case	false	native	throws

Catch	final	new	transient
Char	finally	null	true
Class	float	package	try
Const	for	private	typeof
Continue	function	protected	var
Debugger	goto	public	void
Default	if	return	volatile
Delete	implements	short	while
Do	import	static	with
Double	in	super	

JavaScript- Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<scripttype="text/javascript">
<!--
function functionname(parameter-list)
```

```
{
  statements
}
//-->
</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters −

```
<scripttype="text/javascript">
<!--
function sayHello()
{
        alert("Hello there");
}
//-->
</script>
```

Calling aFunction

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<scripttype="text/javascript">
function sayHello()
{
          document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
<head>
<scripttype="text/javascript">
function sayHello(name, age)
{
        document.write (name +" is "+ age +" years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="sayHello('Zara',7)"value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Thereturn Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.
For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>
<scripttype="text/javascript">
function concatenate(first, last)
{
var full;
        full = first + last;
return full;
}
```

```
function secondFunction()
{
var result;
        result = concatenate('Zara','Ali');
        document.write (result );
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="secondFunction()"value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

JavaScript- Events

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

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

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

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand a relation between Event and JavaScript −

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

```
<html>
<head>
<scripttype="text/javascript">
<!--
```

```
function sayHello(){
            alert("Hello World")
}
//-->
</script>
</head>
<body>
<p>Click the following button and see result</p>
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</form>
</body>
</html>
```

onsubmit Eventtype

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

Example

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

Try the following example.

```
<html>
<head>
<scripttype="text/javascript">
<!--
function validation(){
            all validation goes here
.........
return either true or false
}
//-->
</script>
</head>
<body>
<formmethod="POST"action="t.cgi"onsubmit="return validate()">
       .......
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

onmouseoverandonmouseout

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

```
<html>
<head>
<scripttype="text/javascript">
<!--
function over(){
        document.write ("Mouse Over");
}
function out(){
        document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<divonmouseover="over()"onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
Onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
Onbeforeprint	script	Triggers before the document is printed
Onblur	script	Triggers when the window loses focus
Oncanplay	script	Triggers when media can start play, but might has to stop for buffering

oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
Onchange	script	Triggers when an element changes
Onclick	script	Triggers on a mouse click
Oncontextmenu	script	Triggers when a context menu is triggered
Ondblclick	script	Triggers on a mouse double-click
Ondrag	script	Triggers when an element is dragged
Ondragend	script	Triggers at the end of a drag operation
Ondragenter	script	Triggers when an element has been dragged to a valid drop target
Ondragleave	script	Triggers when an element is being dragged over a valid drop target
Ondragover	script	Triggers at the start of a drag operation
Ondragstart	script	Triggers at the start of a drag operation
Ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
Onemptied	script	Triggers when a media resource element suddenly becomes empty.
Onended	script	Triggers when media has reach the end
Onerror	script	Triggers when an error occur
Onfocus	script	Triggers when the window gets focus
Onformchange	script	Triggers when a form changes
Onforminput	script	Triggers when a form gets user input
Onhaschange	script	Triggers when the document has change
Oninput	script	Triggers when an element gets user input
Oninvalid	script	Triggers when an element is invalid
Onkeydown	script	Triggers when a key is pressed
Onkeypress	script	Triggers when a key is pressed and released

Onkeyup	script	Triggers when a key is released
Onload	script	Triggers when the document loads
Onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
Onloadstart	script	Triggers when the browser starts to load the media data
Onmessage	script	Triggers when the message is triggered
Onmousedown	script	Triggers when a mouse button is pressed
Onmousemove	script	Triggers when the mouse pointer moves
Onmouseout	script	Triggers when the mouse pointer moves out of an element
Onmouseover	script	Triggers when the mouse pointer moves over an element
Onmouseup	script	Triggers when a mouse button is released
Onmousewheel	script	Triggers when the mouse wheel is being rotated
Onoffline	script	Triggers when the document goes offline
Onoine	script	Triggers when the document comes online
Ononline	script	Triggers when the document comes online
Onpagehide	script	Triggers when the window is hidden
Onpageshow	script	Triggers when the window becomes visible
Onpause	script	Triggers when media data is paused
Onplay	script	Triggers when media data is going to start playing
Onplaying	script	Triggers when media data has start playing
Onpopstate	script	Triggers when the window's history changes
Onprogress	script	Triggers when the browser is fetching the media data
Onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes

Onredo	script	Triggers when the document performs a redo
Onresize	script	Triggers when the window is resized
Onscroll	script	Triggers when an element's scrollbar is being scrolled
Onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
Onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
Onselect	script	Triggers when an element is selected
Onstalled	script	Triggers when there is an error in fetching media data
Onstorage	script	Triggers when a document loads
Onsubmit	script	Triggers when a form is submitted
Onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
Ontimeupdate	script	Triggers when media changes its playing position
Onundo	script	Triggers when a document performs an undo
Onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
Onwaiting	script	Triggers when media has stopped playing, but is expected to resume

JavaScript- Document Object Model or DOM

Every web page resides inside a browser window which can be considered as an object.

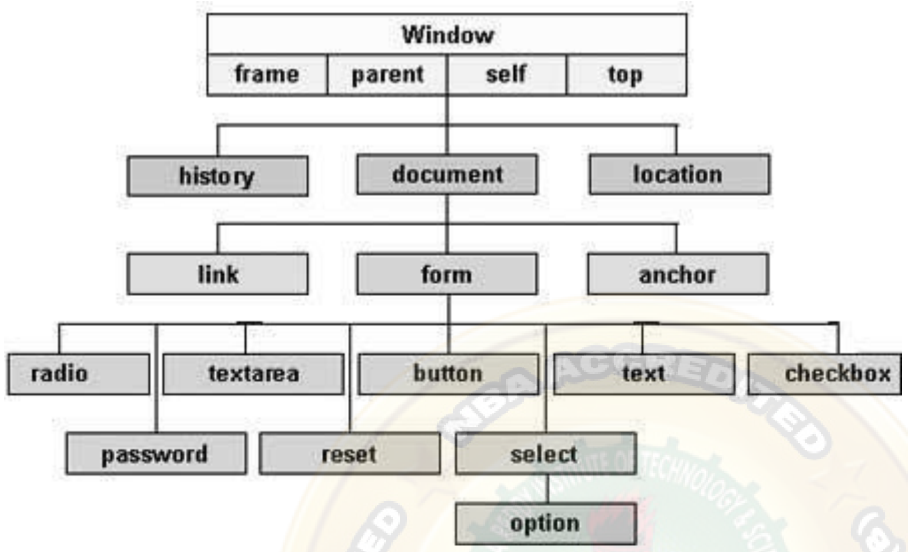
A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** − Top of the hierarchy. It is the outmost element of the objecthierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the formobject.

- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects −



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

- The Legacy DOM − This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- The W3C DOM − This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- The IE4 DOM − This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

DOM compatibility

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

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

JavaScript- Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>
<head>
<title>Form Validation</title>
<scripttype="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>

<body>
<formaction="/cgi-bin/test.cgi"name="myForm"onsubmit="return(validate());">
<tablecellspacing="2"cellpadding="2"border="1">
<tr>
<tdalign="right">Name</td>
<td><inputtype="text"name="Name"/></td>
</tr>
<tr>
<tdalign="right">EMail</td>
<td><inputtype="text"name="EMail"/></td>
</tr>
<tr>
```

```
<tdalign="right">Zip Code</td>
<td><inputtype="text"name="Zip"/></td>
</tr>
<tr>
<tdalign="right">Country</td>
<td>
<selectname="Country">
<optionvalue="-1"selected>[choose yours]</option>
<optionvalue="1">USA</option>
<optionvalue="2">UK</option>
<optionvalue="3">INDIA</option>
</select>
</td>
</tr>
<tr>
<tdalign="right"></td>
<td><inputtype="submit"value="Submit"/></td>
</tr>
</table>
</form>
</body>
</html>
```

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<scripttype="text/javascript">
<!--
// Form validation code will come here.
function validate()
{
if( document.myForm.Name.value =="")
{
        alert("Please provide your name!");
        document.myForm.Name.focus();
returnfalse;
}
if( document.myForm.EMail.value =="")
{
        alert("Please provide your Email!");
```

```
            document.myForm.EMail.focus();
returnfalse;
}

if( document.myForm.Zip.value ==""||
         isNaN( document.myForm.Zip.value )||
         document.myForm.Zip.value.length !=5)
{
            alert("Please provide a zip in the format #####.");
            document.myForm.Zip.focus();
returnfalse;
}
if( document.myForm.Country.value =="-1")
{
            alert("Please provide your country!");
returnfalse;
}
return(true);
}
//-->
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```
<scripttype="text/javascript">
<!--
function validateEmail()
{
var emailID = document.myForm.EMail.value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");
if(atpos <1||( dotpos - atpos <2))
{
            alert("Please enter correct email ID")
            document.myForm.EMail.focus();
returnfalse;
}
return(true);
}
//-->
</script>
```