JavaScript is a scripting language designed and first implemented by Netscape (with help from Sun Microsystems). Javascript was created by Brendan Eich at Netscape in 1995 for the purpose of allowing code in webpages. Netscape first introduced a JavaScript interpreter in Navigator 2. The interpreter was an extra software component in the browser that was capable of interpreting JavaSript source code inside an HTML document. This meant that Web page authors using no more than a simple text editor could place special instructions or programs directly inside Web pages to make them more dynamic and interactive. JavaScript can be used to:

- validate user input in an HTML form before sending the data to a server
- build forms that respond to user input without accessing a server
- change the appearance of HTML documents and dynamically write HTML into separate Windows
- open and close new windows or frames
- manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window
- build small but complete client side programs

Java Script Features

- Java Script is a scripting language and it is not java.
- Java script is interpreter based scripting language.
- Java script is case sensitive.
- Java script is object based language as it provides predefined objects.
- Every statement in java script must be terminated with ;
- Most of the java script control statements syntax is same as syntax of control statements in C language.

Data Types and Variable Declaration

Java script didn't provide any data types for declaring variables and a variable in java script can store any type of value. Hence java script is loosely typed language. You can use a variable directly without declaring it as follows.

X = 10;

You can also declare a variable and then use it and for this use the keyword ***var*** as follows

Var x = 10;

Or

Var x;

X=10;

Writing Java Script

Java script can be written either as **inline java script** or **internal java script** or **external java script.**

Inline Java Script

When java script was written within the html element using attributes related to events of the element then it is called as inline java script.

Java Script Dialog Boxes

Alert() : Alert function of java script is used to give an alert message to the user.

Prompt() : Prompt function of java script is used to prompt for a value from the user.

Confirm() : confirm function of java script is used to get confirmation from user before executing some task.

Example : The following example gives an alert message to the user when user click on the button using inline java script.

```
<!doctype html>

<html>

<form>

<input type="button" value="Click" onclick="alert('Button Clicked')"/>

</form>

</html>
```

Internal Java Script

When java script was written within the <head> section using <script> element then it is called as internal java script. The <script> element syntax is as follows.

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

</script>
```

Within the <head> section java script will be written in the form of functions. To create a function in java script use the keyword ***function*** and it has the following syntax.

Defining Functions Within Scripts

An important part of JavaScript is the ability to create new functions within scripts. These functions are named segments of JavaScript statements. These statements usually have a single purpose, such as performing a complex calculation or verifying the data entered into an HTML form. Functions can be passed copies of variables or references to objects to work with. Just as with built in functions, this is done in the parameter list. JavaScript functions that you define can be placed inside script tags anywhere in the document. However, they should be placed in the head of the document to guarantee they are loaded before being called from script statements in the body of the document. Here is the format for defining a JavaScript function:

```
function functionName(parameter_1, parameter_2, ..  parameter_n)

{

  statement1

  statement2

  statementn

}
```

The keyword function precedes the name of the function. A function name is like a variable name in that it must start with a character and must not be the same as a JavaScript key word. The parameter list is a comma separated list of variable names inside round brackets immediately following the function name. The statements to be executed when the function is called are contained inside the function body and are contained within curly braces.

Here is a function that finds the maximum value of two numbers. Note that the function determines which number is largest and returns the largest value using the return statement. The value returned will take the place of the function in the calling expression - see the example below.

```
 function getMax(n1, n2)

{

    if (n1 < n2)
```

```
    return n2
  else
    return n1
}
```

Example : The following example displays an alert message after prompting name of the user when user click on the button using internal java script.

```
<!doctype html>
<html>
<head>
<script>
Function getname()
{
Name=prompt('Enter Your Name");
Alert('Welcome Mr/Mrs ' + name);
}
</script>
</head>
<form>
<input type="button" value="Click" onclick="getname()"/>
</form>
</html>
```

External Java Script

Writing java script in a separate file with extension .js is called as external java script. For adding the reference of an external java script file to your html page, use ***<script>*** tag with ***src*** attribute as follows

```
<script type="text/javascript" src="filename.js"/>
```

Example : the following example demonstrates how to create and use external java script.

Create a file with name functions.js and write the following java script functions in it.

```
function sum(x,y)
{
return parseInt(x) + parseInt(y);
}
function sub(x,y)
{
return parseInt(x) - parseInt(y);
}
function mul(x,y)
{
return parseInt(x) * parseInt(y);
}
function div(x,y)
{
return parseInt(x) / parseInt(y);
}
```

Create a html page and use the file functions.js as follows

```
<!doctype html>
<html>
<head>
<script src="d:\functions.js">


</script>
```

```
</head>

<body>

<form>

<table>

<tr>

<td> First Number </td>

<td> <input type="text" name="t1"/> </td>

</tr>

<tr>

<td> Second Number </td>

<td> <input type="text" name="t2"/> </td>

</tr>

<tr>

<td> Result </td>

<td> <input type="text" name="t3"/> </td>

</tr>

<tr>

<td colspan="2">

<input type="button" value="sum"
onclick="t3.value=sum(t1.value,t2.value)"/> 

<input type="button" value="sub"
onclick="t3.value=sub(t1.value,t2.value)"/> 

<input type="button" value="mul"
onclick="t3.value=mul(t1.value,t2.value)"/> 

<input type="button" value="div"
onclick="t3.value=div(t1.value,t2.value)"/> 

</td>
```

```
</tr>
</table>
</form>
</body>
</html>
```

Conditional Control Statements

If

```
If(<condition>)
        <stmts>
Else
        <stmts>
```

```
if(<condition>)
        <stmts>
else if(<condition>)
        <stmts>
Else if(<condition>)
        <stmts>
….
Else
        <stmts>
```

Switch

```
Switch(<expression>)
{
        Case <result1> :
                <stmts>
                Break;
        Case <result2>:
                <stmts>
                Break;
```

…..

Default :

<stmts>

}

Ternary Operator (? :)

<var> = (<condition>)?<stmt1>:<stmt2>;

Looping Control Statements

While

<var initialization>

While(<condition>)

{

<stmts>

<Var increment / decrement>

}

For

For(initialization;condition;incr/decr)

{

<stmts>

}

Do while

<variable initialization>

Do

{

<stmts>

<var increment / decrement>

}while(<condition>);

Working With Arrays

An array is a very common data structure - a way of organizing and accessing data. It is supported by almost every computer programming language. Data such as numbers are stored and accessed as though they were in a list. Usually, arrays are created by some kind of declaration that indicates the type of data being stored in each "slot" or "box" in the array and how many "slots" the array should have. JavaScript 1.1, introduced with Navigator 3 included a new prebuilt array object. To create an array you use new Array(n) where n was the number of slots in the array or new Array() omitting the size. For example, using the new keyword Array:

```
mynumbers = new Array(5)

mynumbers[0] = 34

mynumbers[1] = 22

mynumbers[2] = 9

mynumbers[3] = 12

mynumbers[4] = 0


sum = 0;

for (i=0; i<mynumbers.length; i++)

{

        sum = sum + mynumbers[i];

}

alert(sum)
```

The new array object had the advantage that you could create the array and add values at the same time:

mynumbers = new Array(34, 22, 9, 12, 0)

This made it possible to do in one line what it took six lines to do in the previous example.

The one problem with this was that ***mnumbers = new Array(2, 1)*** creates an array with two values in it but ***mynumbers = new Array(2)*** creates an array with two empty slots. Logically, it should create one slot with the number 2 in it.

Example : The following example demonstrates how to work with one dimensional arrays

```
<html>
<head>
<script type="text/javascript">
var A=new Array(5);
for(i=0;i<5;i++)
{
A[i]=eval(prompt("Enter An Integer"));
}
for(i=0;i<5;i++)
{
document.write("<h1>" + A[i] + "</h1>");
}
</script>
</head>
<body>
</body>
</html>
```

Two Dimesional Arrays

Two dimensional arrays in java script are declared using the following syntax.

```
var x = new Array(10);
 for (var i = 0; i < 10; i++)
 {
```

```
   x[i] = new Array(20);
 }
 x[5][12] = 3.0;
```

Example : The following example demonstrates how to use two dimensional arrays.

```
<html>
<head>
<script type="text/javascript">
var A = new Array(3);
for (var i = 0; i < 3; i++)
{
A[i] = new Array(3);
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
A[i][j]=eval(prompt("Enter An Integer"));
}
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
document.write("<font size=7>" + A[i][j] +

"  </font>");
```

```
}
document.write("<br/>");
}
</script>
</head>
<body>
</body>
</html>
```

Example : The following example demonstrates how to use Array ***Concat()*** Method. This method concats the elements of one array at the end of another array and returns an array.

```
<html>
<head>
<script type="text/javascript">
var A=new Array(23,45,17,31,59);
var B=new Array("a","b","c");
var C=B.concat(A);
document.write("<h1>" + C.toString() + "</h1>");
</script>
</head>
<body>
</body>
</html>
```

Example : The following example demonstrates how to use Array ***Sort() and Reverse()*** Methods. Sort() methods sorts the elements in array and reverse() methods reverses the elements.

```
<html>
<head>
```

```
<script type="text/javascript">
function compare(a,b)
{
return a-b;
}
var A=new Array(23,45,17,31,59,130);
var B=A.sort(compare);
document.write("<h1>" + B.toString() + "</h1>");
document.write("<h1>" + B.reverse() + "</h1>");
</script>
</head>
<body>
</body>
</html>
```

Example : The following example demonstrates how to use Array ***Slice()*** Method. Slice method will extract specified number of elements starting from specified index without deleting them from array.

```
<html>
<head>
<script type="text/javascript">
var A=new Array(10,20,30,40,50,60,70);
var B=A.slice(1,4);
document.write("<h1>" + A.toString() + "</h1>");
document.write("<h1>" + B.toString() + "</h1>");
</script>
</head>
<body>
```

```
</body>

</html>
```

Example : The following example demonstrates how to use Array ***Splice()*** Method. Splice method will extract specified number of elements starting from specified index and deletes them from array.

```
<html>

<head>

<script type="text/javascript">

var A=new Array(10,20,30,40,50,60,70);

var B=A.splice(1,4);

document.write("<h1>" + A.toString() + "</h1>");

document.write("<h1>" + B.toString() + "</h1>");

</script>

</head>

<body>

</body>

</html>
```

Example : The following example demonstrates how to use Array ***Splice()*** Method. Splice method can extract specified number of elements starting from specified index and replace them with other elements.

```
<html>

<head>

<script type="text/javascript">

var A=new Array(10,20,30,40,50,60,70);

var B=A.splice(1,4,1,2,3,4);

document.write("<h1>" + A.toString() + "</h1>");

document.write("<h1>" + B.toString() + "</h1>");
```

```
</script>

</head>

<body>

</body>

</html>
```

Example : The following example demonstrates how to use Array ***Push() and Pop()*** Methods. Push method pushes element at the top and pop elements pops the top element.

```
<html>

<head>

<script type="text/javascript">

var A=new Array(10,20,30,40,50);

A.push(eval(prompt("Enter  Value to push")));

document.write("<h1>" + A.toString() + "</h1>");

A.push(eval(prompt("Enter  Value to push")));

document.write("<h1>" + A.toString() + "</h1>");

var p=A.pop();

document.write("<h1>Element Popped Is " + p + "</h1>");

var p=A.pop();

document.write("<h1>Element Popped Is " + p + "</h1>");

document.write("<h1>" + A.toString() + "</h1>");

</script>

</head>

<body>

</body>

</html>
```

Example : The following example demonstrates how to use Array ***Sort()*** method for sorting strings.

```
<html>
<head>
<script type="text/javascript">
function compare(a,b)
{return a.localeCompare(b);
}
var A=new Array('Microsoft','Adobe','Oracle','Sun','IBM');
document.write("<h1>" + A.sort(compare) + "</h1>");
</script>
</head>
<body>
</body>
</html>
```

Validations

One main purpose of java script is performing validations. The following examples demonstrates how to perform various validations in java script.

Example : The following example demonstrates how to restrict a text box to accept only digits.

```
<html>
<head>
<script type="text/javascript">
function fn_validateNumeric(thi,dec)
{
```

```
/* verifying the character pressed is a digit */

if (((event.keyCode < 48) || (event.keyCode > 57)) &&

(event.keyCode != 46))

{

event.returnValue = false;

}

/* determining whether or not allow decimal point(.) if second argument is "n" then decimal point not allowed. else part will check whether the decimal point (.) is pressed second time and if it is second time then not allowed*/

if(dec=="n" && event.keyCode == 46)

{

event.returnValue = false;

}

else

{

if(event.keyCode == 46 && thi.value.indexOf(".")>=0)

{

event.returnValue = false;

}

}

}

/* Ascii values of 0-9 are 48 to 57 and ascii value of "." is 46. "This" refers to the control that causes the event.  else condition is to restrict "." more than once  */

</script>

</head>

<body>

<input type="text" onkeypress=
```

```
"fn_validateNumeric(this,'y')" name="t1"/>

</body>

</html>
```

Example : The following example demonstrates how to restrict a text box to accept only alphabets.

```
<html>

<head>

<script type="text/javascript">

function fn_validateAlpha(thi)

{

if (((event.keyCode < 65) || (event.keyCode > 90)) && ((event.keyCode <97) || (event.keyCode>122)))

{

event.returnValue = false;

}

}

</script>

</head>

<body>

<input type="text" onkeypress="fn_validateAlpha(this)" name="t1"/>

</body>

</html>
```

Example : The following example demonstrates how to automatically convert the letters typed in a text box to uppercase.

```
<html>

<head>

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

```
function Convert_Capital(thi)
{
if ((event.keyCode >=97) && (event.keyCode<=122))
[
event.keyCode=event.keyCode-32;
}
}
</script>
</head>
<body>
<input type="text" onkeypress="Convert_Capital(this)" name="t1"/>
</body>
</html>
```

Example : The following example demonstrates how to perform various validations like textbox is not blank , password is minimum 8 characters, password and confirm password are same and email id is in correct format.

```
<html>
<head>
<script type="text/javascript">
function emailcheck()
{
var str=f1.txtemail.value;
var at="@";
var dot=".";
var atpos=str.indexOf(at);
var len=str.length;
var dotpos=str.indexOf(dot);
```

```
if (atpos==-1 || atpos==0 || atpos==len)
{
alert("Invalid E-mail ID");
return false;
}
if (dotpos==-1 || dotpos==0 || dotpos==len)
{
alert("Invalid E-mail ID");
return false;
}
if (str.indexOf(at,(atpos+1))!=-1)
{
alert("Invalid E-mail ID");
return false;
}
/* checking immediate before character of @ or immediate next character of @ is .
*/
if (str.substring(atpos-1,atpos)==dot || str.substring(atpos+1,atpos+2)==dot)
{
alert("Invalid E-mail ID");
return false;
}
/* Checking . exists next to @ symbol or not */
if (str.indexOf(dot,(atpos+2))==-1)
{
```

```
alert("Invalid E-mail ID");
return false;
}
if (str.indexOf(" ")!=-1)
{
alert("Invalid E-mail ID");
return false;
}
return true;
}
function isvalid()
{
if(f1.txtuname.value=="")
{
alert("You Must Enter User Name");
f1.txtuname.focus();
return false;
}
if(f1.txtpwd1.value=="" || f1.txtpwd2.value=="")
{
alert("You Must Enter Password And Re Enter Password");
f1.txtpwd1.focus();
return false;
}
if(f1.txtpwd1.value.length<8)
{
```

```
alert("Password Must Be Minimum 8 characters");
f1.txtpwd1.value="";
f1.txtpwd2.value="";
f1.txtpwd1.focus();
return false;
}
if(f1.txtpwd1.value!=f1.txtpwd2.value)
{
alert("Passsword and Re-Enter Password Must Be Same");
f1.txtpwd1.value="";
f1.txtpwd2.value="";
f1.txtpwd1.focus();
return false;
}
if(f1.txtemail.value=="")
{
alert("You Must Enter Email Id");
f1.txtemail.focus();
return false;
}
if(emailcheck()==false)
{
f1.txtemail.value="";
f1.txtemail.focus();
return false;
}
```

```
return true;
}
</script>
</head>
<body>
<form id="f1">
<table>
<tr>
<td> Enter User Name </td>
<td><input type="text" name="txtuname" /></td>
</tr>
<tr>
<td> Enter Password </td>
<td><input type="password" name="txtpwd1"/></td>
</tr>
<tr>
<td> Re-Enter Password </td>
<td><input type="password" name="txtpwd2"/></td>
</tr>
<tr>
<td> Enter Email id </td>
<td><input type="text" name="txtemail"/></td>
</tr>
<tr>
<td colspan="2" align="right"><input type="button" name="btnregister" value="Register" onclick="isvalid()"/></td>
```

</tr>

</table>

</form>

</body>

</html>

Cookies

Web Browser and Server use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after completing many pages. But how to maintain user's session information across all the web pages. In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

Expires : The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.

Domain : The domain name of your site.

Path : The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.

Secure : If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.

Name=Value : Cookies are set and retrieved in the form of key and value pairs.

Cookies were originally designed for CGI programming and cookies' data is automatically transmitted between the web browser and web server, so CGI scripts on the server can read and write cookie values that are stored on the client. JavaScript can also manipulate cookies using the cookie property of the Document object. JavaScript can read, create, modify, and delete the cookie or cookies that apply to the current web page.

Storing Cookies

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this:

Syntax:

document.cookie = "key1=value1;key2=value2;expires=date";

Here expires attribute is option. If you provide this attribute with a valid date or time then cookie will expire at the given date or time and after that cookies' value will not be accessible.

Note: Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript escape() function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding unescape() function when you read the cookie value.

Example : The following example demonstrates how to work with cookies for storing user id and password in cookies in a login form.

```
<!doctype html>

<html>

<head>

<script>

function loginclick()

{

var c=document.getElementsByName("chkremember")[0];

if(c.checked)

{

setCookie("uid1",f1.txtuid.value,7);

}

}

function getCookie(c_name)

{
```

```
var i,x,y,ARRcookies=document.cookie.split(";");

for (i=0;i<ARRcookies.length;i++)

{

x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));

y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);

if (x==c_name)

{

return unescape(y);

}

}

}

function setCookie(c_name,value,exdays)

{

var exdate=new Date();

exdate.setDate(exdate.getDate() + exdays);

var c_value= escape(value) + ((exdays==null) ? "" : ";
expires="+exdate.toUTCString());

document.cookie=c_name + "=" + c_value;

}

function checkCookie()

{

var username=getCookie("uid1");

alert(username);

if (username!=null && username!="")

{

return username;
```

```
}

}

</script>

</head>

<body onload="f1.txtuid.value=checkCookie()">

<form id="f1">

<table>

<tr>

<td> User Name </td>

<td> <input type="text" name="txtuid"/> </td>

</tr>

<tr>

<td> Password </td>

<td> <input type="password" name="txtpwd"/> </td>

</tr>

<tr>

<td colspan="2"> <input type="checkbox" name="chkremember"/> Remember Me
</td>

</tr>

<tr>

<td Colspan="2" align="right"> <input type="button" name="btnlogin"
value="login" onclick="loginclick()"/> </td>

</tr>

</table>

</form>

</body>

</html>
```

DOM (Document Object Model) "Window" Object

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

DOM (Document Object Model) "Window" Object Properties

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the statusbar of a window
document	Returns the Document object for the window (See Document object)
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window (See History object)
innerHeight	Sets or returns the inner height of a window's content area
innerWidth	Sets or returns the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window
location	Returns the Location object for the window (See Location object)
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window (See Navigator object)
opener	Returns a reference to the window that created the window

outerHeight	Sets or returns the outer height of a window, including toolbars/scrollbars
outerWidth	Sets or returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window (See Screen object)
screenLeft	Returns the x coordinate of the window relative to the screen
screenTop	Returns the y coordinate of the window relative to the screen
screenX	Returns the x coordinate of the window relative to the screen
screenY	Returns the y coordinate of the window relative to the screen
self	Returns the current window
status	Sets the text in the statusbar of a window
top	Returns the topmost browser window

DOM (Document Object Model) "Window" Object Methods

Method	Description
alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
createPopup()	Creates a pop-up window
focus()	Sets focus to the current window
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height

scroll()	
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

DOM (Document Object Model) "Document" Object

Each HTML document loaded into a browser window becomes a Document object. The Document object provides access to all HTML elements in a page, from within a script. The Document object is also part of the Window object, and can be accessed through the ***window.document*** property.

DOM (Document Object Model) "Document" Object Properties

Property	Description
anchors	Returns a collection of all the anchors in the document
applets	Returns a collection of all the applets in the document
body	Returns the body element of the document
cookie	Returns all name/value pairs of cookies in the document
documentMode	Returns the mode used by the browser to render the document
domain	Returns the domain name of the server that loaded the document
forms	Returns a collection of all the forms in the document

images	Returns a collection of all the images in the document
lastModified	Returns the date and time the document was last modified
links	Returns a collection of all the links in the document
readyState	Returns the (loading) status of the document
referrer	Returns the URL of the document that loaded the current document
title	Sets or returns the title of the document
URL	Returns the full URL of the document

DOM (Document Object Model) "Document" Object Methods

Method	Description
close()	Closes the output stream previously opened with document.open()
getElementsByName()	Accesses all elements with a specified name
open()	Opens an output stream to collect the output from document.write() or document.writeln()
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

DOM (Document Object Model) "Navigator" Object

The navigator object contains information about the browser from where user is accessing your page.

DOM (Document Object Model) "Navigator" Object Properties

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
onLine	Boolean, returns *true* if the browser is on line, otherwise *false*.
platform	Returns for which platform the browser is compiled
userAgent	Returns the user-agent header sent by the browser to the server

DOM (Document Object Model) "History" Object

The history object contains the URLs visited by the user (within a browser window). The history object is part of the window object and is accessed through the ***window.history*** property.

DOM (Document Object Model) "History" Object Properties

Property	Description
length	Returns the number of URLs in the history list

DOM (Document Object Model) "History" Object Methods

Method	Description
back()	Loads the previous URL in the history list

forward()	Loads the next URL in the history list
go()	Loads a specific URL from the history list

Example : The following example demonstrates how to use ***open()*** method of ***window*** object and ***opener*** *property of* ***window*** object and ***document*** object's ***bgcolor*** property. First open parent.html in browser that will automatically open child.html in a new window or tab and then click on buttons in child.html to change background color of parent.html.

Parent.Html

```
<html>
<head>
<script type="text/javascript">
var ch=window.open("child.html","win1","width=300,height=300");
ch.document.bgColor="purple";
</script>
</head>
</html>
```

Child.Html

```
<html>
<head>
<title>Secondary window</title>
</head>
<body >
<center>
<form>
<input type="button" onClick=
"window.opener.document.bgColor='yellow'" value="yellow"> 
```

```
<input type="button" onClick=
"window.opener.document.bgColor='lightgreen'" value="lightgreen"> 
<input type="button" onClick=
"window.opener.document.bgColor='white'" value="white">
</form>
</center>
</body>
</html>
```

Example : The following example demonstrates various properties of window object that can be specified using third argument of ***open*** method of ***window*** object.

```
<html>
<head>
<script type="text/javascript">
function newwindow()
{
var win=window.open("password.html","Win1", "width=300,height=300,resizable,
toolbar,left=100,top=100");
}
</script>
</head>
<body onload="newwindow()">
</body>
</html>
```

Example : The following example demonstrates the properties of ***navigator*** object

```
<html>
```

```
<body>

<script type="text/javascript">

document.write("<h3> Your Browser is : " + navigator.appName + "</h3>");

document.write("<h3> Your Browser Version is : " + navigator.appVersion + "</h3>");

document.write("<h3> Your Platform is : " + navigator.platform + "</h3>");

</script>

</body>

</html>
```

Example : The following example demonstrates the use ***setinterval()***/ and ***clearinterval()*** methods of ***window*** object.

```
<html>

<head>

<script type="text/javascript">

var id,i=0;

var colors=new Array("red","green","blue","yellow","white");

function FStart()

{

id=setInterval("ChangeColor()",1000);

}

function FStop()

{

clearInterval(id);

}

function ChangeColor()

{

document.bgColor=colors[i];
```

```
i++;
if(i==5)
i=0;
}
</script>
</head>
<body>
<input type="button" value="Start" onclick="FStart()"/>
<input type="button" value="Stop" onclick="FStop()"/>
</body>
</html>
```

Example : The following example demonstrates the use of ***history*** object.

First.Html

```
<!doctype html>
<html>
<body>
<a href="history.html">
<img src="e:\images\bird.jpg" width="200" height="200"/></a>
</body>
</html>
```

History.Html

```
<!doctype html>
<html>
<body>
<a href="Next.html"><img src="e:\images\forest flowers.jpg" width="200"
height="200"/></a><br/><br/>
```

```
<input type="button" value="Go Back"
onclick="window.history.back()"/>  

<input type="button" value="Go Next"
onclick="window.history.forward()"/>  

</body>

</html>
```

Next.html

```
<!doctype html>

<html>

<body>

<img src="e:\images\Creek.jpg"

width="200" height="200"/>

</body>

</html>
```

In the above example first run first.html from browser and click on the image that displays history.html, click on the image in history.html to display next.html and then click on browser back button to move back to the history.html. now within the history.html click on “go back” button to go to the page first.html and click on “go next” button to go to the page next.html.