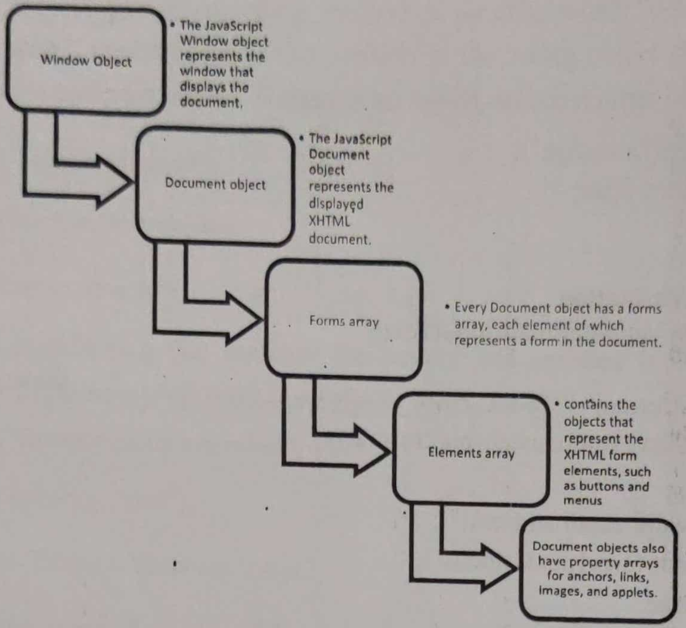
Unit 3

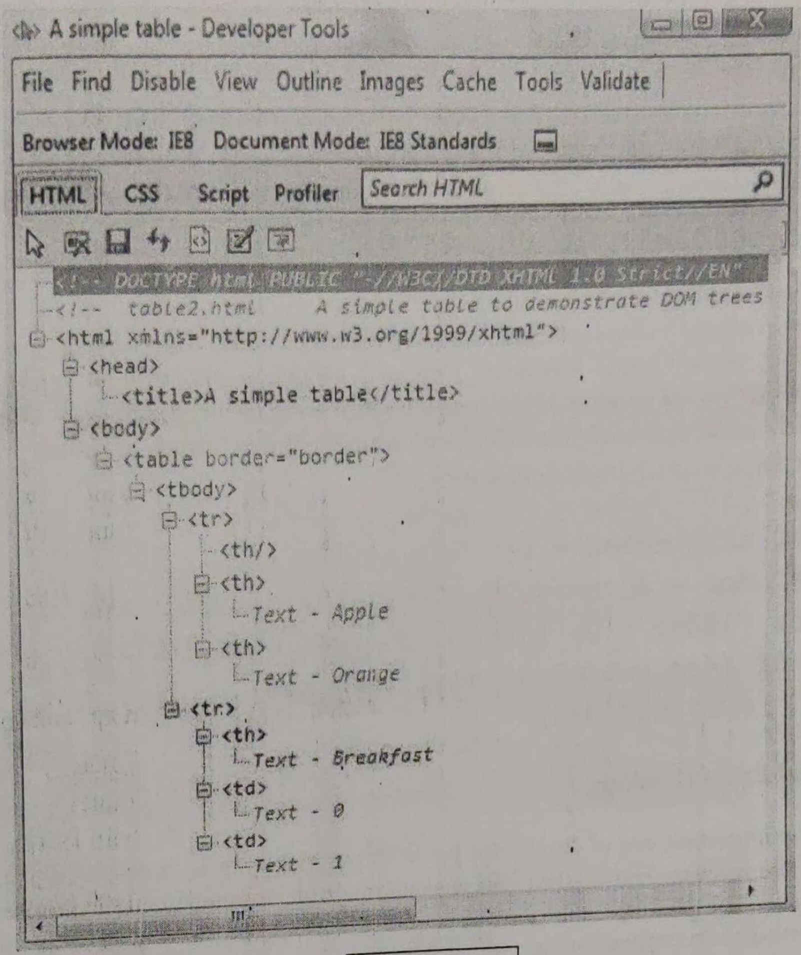
JavaScript and XHTML Documents

The JavaScript Execution Environment

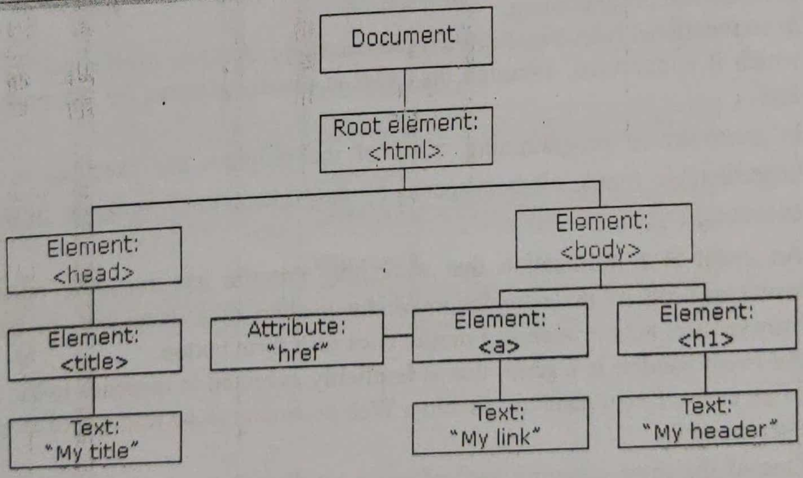

Window Object
• The JavaScript Window object represents the window that displays the document.
Document object
• The JavaScript Document object represents the displayed XHTML document.
Forms array
• Every Document object has a forms array, each element of which represents a form in the document.
Elements array
• contains the objects that represent the XHTML form elements, such as buttons and menus
Document objects also have property arrays for anchors, links, images, and applets.

Document Object Model

- The DOM is an application programming interface (API) that defines an interface between XHTML documents and application programs.
- It is an abstract model because it must apply to a variety of application programming languages.
- Each language that interfaces with the DOM must define a binding to that interface.
- DOM specification consists of a collection of interfaces, including one for each document tree node type.
- Interfaces define the objects, methods, and properties that are associated with their respective node types.
- Using DOM users can write code in programming languages to create documents, move around in their structures, and change, add, or delete elements and their content.
- Documents in the DOM have a treelike structure as described in the following example.


A simple table - Developer Tools
File Find Disable View Outline Images Cache Tools Validate
Browser Mode: IE8 Document Mode: IE8 Standards
HTML CSS Script Profiler
Search HTML

```
<!-- DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
<!--   table2.html     A simple table to demonstrate DOM trees
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>A simple table</title>
  <body>
    <table border="border">
      <tbody>
        <tr>
          <th/>
          <th>
            Text - Apple
          <th>
            Text - Orange
        <tr>
          <th>
            Text - Breakfast
          <td>
            Text - 0
          <td>
            Text - 1
```



Element Access in JavaScript

The elements of an XHTML document have corresponding objects that are visible to an embedded JavaScript script. The addresses of these objects are required, both by the event handling discussed in this chapter and by the code for making dynamic changes to documents. Ways to access HTML elements in JavaScript:

1. Using DOM Address: Use the forms and elements arrays of the Document object, which is referenced through the document property of the Window object.
   E.g., var dom = document.forms[0].elements[0];
2. Using element name property: Another approach to DOM addressing is to use element names. E.g., var dom = document.myForm.txtUserName;
3. Using getElementById(): Element addressing using the JavaScript method getElementById.
   E.g., var dom = document.getElementById("txtUserName");
4. Using implicit arrays associated with each checkbox and radio button group: Every such group has an array, which has the same name as the group name, that stores the DOM addresses of the individual buttons in the group. These arrays are properties of the form in which the buttons appear.
   E.g.,
   ```
   var numChecked = 0;
   var dom = document.getElementById("vehicleGroup");
   for (index = 0; index < dom.vehicles.length; index++)
     if (dom.vehicles[index].checked)
       numChecked++;
   ```

Events and Event Handling

- One important use of JavaScript for Web programming is to detect certain activities of the browser and the browser user and provide computation when those activities occur. These computations are specified with a special form of programming called event-driven programming.
- In conventional (non-event-driven) programming, the code itself specifies the order in which it is executed, although the order is usually affected by the program's input data.
- In event-driven programming, parts of the program are executed at completely unpredictable times, often triggered by user interactions with the program that is executing.
- An event is a notification that something specific has occurred, either with the browser, such as the completion of the loading of a document, or because of a browser user action, such as a mouse click on a form button.
- An event handler is a script that is implicitly executed in response to the appearance of an event. Event handlers enable a Web document to be responsive to browser and user activities.
- One of the most common uses of event handlers is to check for simple errors and omissions in user input to the elements of a form, either when they are changed or when the form is submitted.

- This kind of checking saves the time of sending incorrect form data to the server.
- Because events are JavaScript objects, their names are case sensitive. The names of all event objects have only lowercase letters.
- Events are created by activities associated with specific XHTML elements.
- The process of connecting an event handler to an event is called registration.
- There are two distinct approaches to event handler registration, one that assigns tag attributes and one that assigns handler addresses to object properties.

Events, Attributes, and Tags

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Attribute	Tag	Description
onblur	<a>	The link loses the input focus
	<button>	The button loses the input focus
	<input>	The input element loses the input focus
	<textarea>	The text area loses the input focus
	<select>	The selection element loses the input focus
onchange	<input>	The input element is changed and loses the input focus
	<textarea>	The text area is changed and loses the input focus
	<select>	The selection element is changed and loses the input focus
onclick	<a>	The user clicks on the link
	<input>	The input element is clicked
ondblclick	Most elements	The user double-clicks the left mouse button
onfocus	<a>	The link acquires the input focus
	<input>	The input element receives the input focus
	<textarea>	A text area receives the input focus
	<select>	A selection element receives the input focus
onkeydown	<body>, form elements	A key is pressed down
onkeypress	<body>, form elements	A key is pressed down and released
onkeyup	<body>, form elements	A key is released
onload	<body>	The document is finished loading
onmousedown	Most elements	The user clicks the left mouse button
onmousemove	Most elements	The user moves the mouse cursor within the element
onmouseout	Most elements	The mouse cursor is moved away from being over the element
onmouseover	Most elements	The mouse cursor is moved over the element
onmouseup	Most elements	The left mouse button is unclicked
onreset	<form>	The reset button is clicked
onselect	<input>	Any text in the content of the element is selected
	<textarea>	Any text in the content of the element is selected
onsubmit	<form>	The *Submit* button is pressed
onunload	<body>	The user exits the document

As mentioned previously, there are two ways to register an event handler in the DOM 0 event model. One of these is by assigning the event handler script to an event tag attribute, as in the following example:

```
<input type = "button" id = "myButton"
       onclick = "alert('You clicked my button!');" />
```

In many cases, the handler consists of more than a single statement. In these cases, often a function is used and the literal string value of the attribute is the call to the function. Consider the example of a button element:

```
<input type = "button" id = "myButton"
       onclick = "myButtonHandler();" />
```

An event handler function could also be registered by assigning its name to the associated event property on the button object, as in the following example:

```
document.getElementById("myButton").onclick =
                                    myButtonHandler;
```

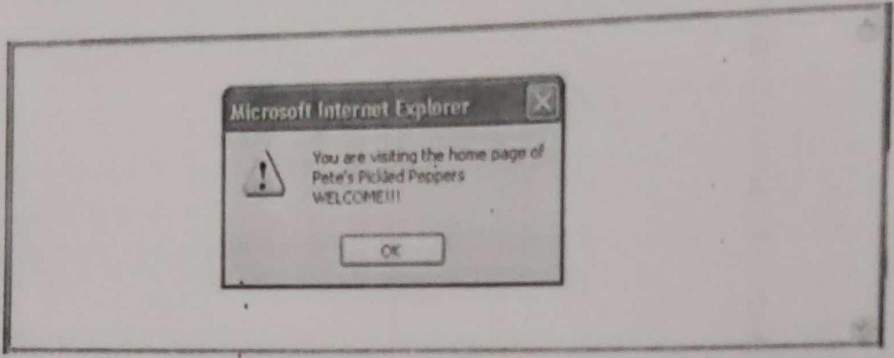
Handling Events from Body Elements

The events most often created by body elements are load and unload. As our first example of event handling, we consider the simple case of producing an alert message when the body of the document has been loaded. In this case, we use the onload attribute of <body> to specify the event handler:

```
<?xml version = "1.0"  encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">


<!-- load.html
     A document for load.js
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> load.html </title>
    <script type = "text/javascript"  src = "load.js" >
    </script>
  </head>
  <body onload="load_greeting();">
    <p />
  </body>
</html>
```

```
// load.js
//   An example to illustrate the load event

// The onload event handler
function load_greeting () {
  alert("You are visiting the home page of \n" +
        "Pete's Pickled Peppers \n" + "WELCOME!!!");

}
```

Output:



```
You are visiting the home page of
Pete's Pickled Peppers
WELCOME!!!
```

Handling Events from Button Elements

Buttons in a Web document provide an effective way to collect simple input from the browser user. The most commonly used event created by button actions is click.

E.g., <input type = "button" value = "Total Cost" onclick = "computeCost();"/>

Handling Events from Text Box and Password Elements

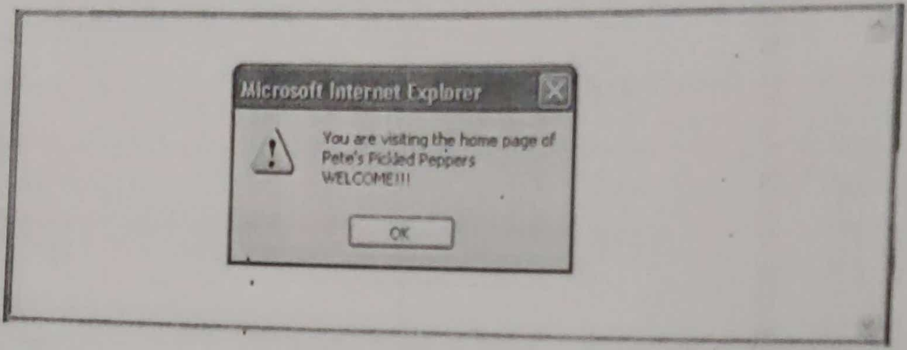
Text boxes and passwords can create four different events: blur, focus, change, and select.

- Blur: The blur event fires when an element has lost focus.
- Focus: The focus event fires when an element has received focus.
- Change: The change event occurs when the value of an element has been changed.
- Select: The select event fires when some text has been selected.

```
//pswd_chk.html

<html> <head> <title>Password Checking</title>
 <script type = "text/javascript">
function chkPass( )
{
var init=document.getElementById("initial");
var fin=document.getElementById("final");
```

```
// load.js
//   An example to illustrate the load event

// The onload event handler
function load_greeting () {
  alert("You are visiting the home page of \n" +
        "Pete's Pickled Peppers \n" + "WELCOME!!!");
}
```

Output:



```
You are visiting the home page of
Pete's Pickled Peppers
WELCOME!!!
```

Handling Events from Button Elements

Buttons in a Web document provide an effective way to collect simple input from the browser user. The most commonly used event created by button actions is click.

E.g., <input type = "button" value = "Total Cost" onclick = "computeCost();"/>

Handling Events from Text Box and Password Elements

Text boxes and passwords can create four different events: blur, focus, change, and select.

- Blur: The blur event fires when an element has lost focus.
- Focus: The focus event fires when an element has received focus.
- Change: The change event occurs when the value of an element has been changed.
- Select: The select event fires when some text has been selected.

```
//pswd_chk.html

<html> <head> <title>Password Checking</title>
 <script type = "text/javascript">
function chkPass( )
{
var init=document.getElementById("initial");
var fin=document.getElementById("final");
if(init.value=="")
{
        alert("You did not enter a Password\n" + "Please enter atleast now");
        init.focus( );
        return false;
}
if(init.value!=fin.value)
{
        alert("The passwords you entered do not match.... Try Again");
        init.focus( );
        init.select( );
        return false; }
        Else
        return true;
}

</script>
</head>
<body>
<h3>Password Input</h3>
<form id="myForm" action=" "> <p> <label>Your Password: <input type="password"
id="initial" size="10"/></label><br/><br/>
<label>Verify Password: <input type="password" id="final" size="10"/></label><br/><br/>
<input type="reset" name="Reset"/> <input type="submit" name="Submit" onclick="
chkPass( );"/> </p>
</form>
</body>
</html>
```

Event Names

Input Events

onblur - When a user leaves an input field

onchange - When a user changes the content of an input field

onchange - When a user selects a dropdown value

onfocus - When an input field gets focus

onselect - When input text is selected

onsubmit - When a user clicks the submit button

onreset - When a user clicks the reset button

onkeydown - When a user is pressing/holding down a key

onkeypress - When a user is pressing/holding down a key

onkeyup - When the user releases a key

onkeyup - When the user releases a key

onkeydown vs onkeyup - Both

Mouse Events

onmouseover/onmouseout - When the mouse passes over an element

onmousedown/onmouseup - When pressing/releasing a mouse button

onmousedown - When mouse is clicked: Alert which element

onmousedown - When mouse is clicked: Alert which button

onmousemove/onmouseout - When moving the mouse pointer over/out of an image

onmouseover/onmouseout - When moving the mouse over/out of an image

onmouseover an image map

Click Events

Acting to the onclick event

onclick - When button is clicked

ondblclick - When a text is double-clicked

Load Events

onload - When the page has been loaded

onload - When an image has been loaded

onerror - When an error occurs when loading an image

onunload - When the browser closes the document

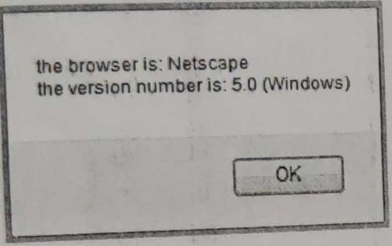
onresize - When the browser window is resized

The navigator Object

The navigator object indicates which browser is being used to view the XHTML document. The browser's name is stored in the appName property of the object. The version of the browser is stored in the appVersion property of the object.

```
<html>
<head>
<title>Navigator</title>
<script type = "text/javascript" >
function navProperties() { alert("the browser is: " + navigator.appName + "\n" + "the version number is: " + navigator.appVersion + "\n"); }

</script>
</head>
<body onload = "navProperties()">
</body> </html>
```

Output:



```
the browser is: Netscape
the version number is: 5.0 (Windows)
```

DOM Tree Traversal and Modification

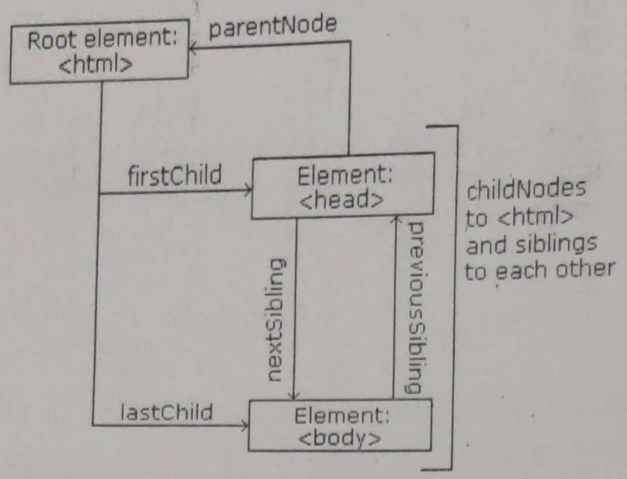
With the HTML DOM, you can navigate the node tree using node relationships.

Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent



```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```

From the HTML above you can read:

- **<html> is the root node**
- **<html> has no parents**
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>

and:

- <head> has one child: <title>
- <title> has one child (a text node): "DOM Tutorial"
- <body> has two children: <h1> and <p>
- <h1> has one child: "DOM Lesson one"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling

The value of the text node can be accessed by the node's InnerHTML property:,

E.g. <title id="demo">DOM Tutorial</title>
For the above E.g, text can be accessed as follows
var myTitle = document.getElementById("demo").innerHTML;
Example to access child nodes:

```
<html>
<body>
 <div>Begin</div>

 <ul>
  <li>Information</li>
 </ul>

 <div>End</div>

 <script>
  for (let i = 0; i < document.body.childNodes.length; i++) {
   alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT
  }
 </script>
 ...more stuff...
</body>
</html>
```

Dom tree modification

A number of methods allow JavaScript code to modify an existing DOM tree structure. The **insertBefore(newChild, refChild) method places the newChild node before the refChild node. The replaceChild(newChild, oldChild) method replaces the oldChild node with the newChild** node. The removeChild(oldChild) method removes the specified node from the DOM structure.

The appendChild(newChild) method adds the given node to the end of the list of siblings of the node through which it is called.

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example:

```
<!DOCTYPE html>
<html>
<body>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
</body>
</html>
```

Creating new HTML Elements - insertBefore()

The appendChild() method in the previous example, appended the new element as the last child of the parent. If you don't want that you can use the insertBefore() method:

Example:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
```

```
element.insertBefore(para, child);
</script>
```

Replacing HTML Elements

To replace an element to the HTML DOM, use the replaceChild() method:

Example:

```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```