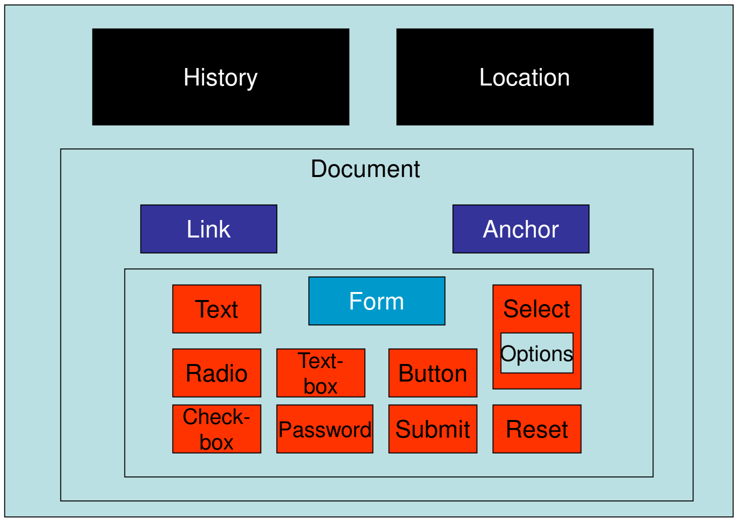
Web Programming

Lecture 4 – JavaScript and HTML Documents

The JavaScript Execution Environment

- Browsers display XHTML documents in a window, which is an object in its own right. JavaScript has a **`Window`** object that represents the window displaying the document.
- All JavaScripts can see the various properties of the **`Window`** object.
- Every **`Window`** has a **`Document`** object that represents the display XHTML documents, which has properties of its own.

`Window` Object



The Document Object Model

- DOM (***D***ocument ***O***bject ***M***odel) is an API (Application Programming Interface) that defines an interface between HTML documents and application program.
- It is an abstract model that applies to a variety of programming language.
- Essentially, the various structures in an HTML document that are marked up with tags are considered objects, complete with properties and methods.
- The attributes of an HTML tag corresponds to a property for the corresponding object.

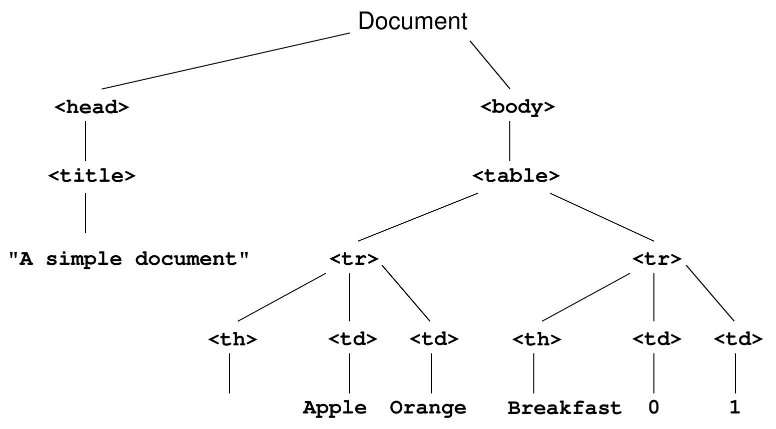
A table in XHTML

```
<!DOCTYPE html>
<!-- table2.html
     A simple table to demonstrate DOM trees
     -->
<html lang = "en">
  <head> <title> A simple document </title>
    <meta charset = "utf-8" />
  </head>
  <body>
    <table>
      <tr>
        <th> </th>
        <td> Apple </td>
        <td> Orange </td>
      </tr>
```

```
      <tr>
        <th> Breakfast </th>
        <td> 1 </td>
        <td> 0 </td>
      </tr>

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

DOM Structure of the Document



DOM – An Example

- The following tag:

```
<input type = "text"
       name = "address">
```

corresponds to an object, which has two properties **`type`** (which is **`text`**) and **`name`** (which is **`address`**).

Element Access in JavaScript

- Manipulating elements in an HTML document requires that you have the address of the corresponding object.
- This can be handled in several different ways, depending on which version of DOM to which you seek to conform.
- The address of a JavaScript object associated with an HTML element is called its ***DOM address***.

Sample XHTML document

```
<html lang = "en">
  <head> <title> Access to form elements </title>
    <meta charset = "utf-8" />
  </head>

  <body>
    <form name = "myForm" action = "">
      <input type = "button" name = "turnItOn">
    </form>
  </body>
</html>
```

Getting the DOM Address

- We can use the forms and elements arrays:
  `var dom = document.forms[0].elements[0];`

  If another button were added before turnItOn, the address could no longer be accessed this way.
- We can use the name given to the item (this requires that every enclosing item (up to but not including body) must have a name:
  `var dom = document.myForm.turnItOn;`

  The problem is that XHTML 1.1 does not allow forms to have a name. This causes a validation problem.

`getElementById`

- A better way involves using getElementById:
  `var dom = document.getElementById("turnItOn");`
- Since ids are useful for DOM addressing and form processing frequently requires name, it is not unusual for form elements to have both set to the same value.
- Since buttons in a group of checkboxes often share a name and a group of radio buttons will share a name, this won't work in finding their DOM addresses.

Checkboxes

```
<form id = "vehicleGroup">
  <input type = "checkbox name = "vehicles"
         value = "car" /> Car
  <input type = "checkbox name = "vehicles"
         value = "truck" /> Truck
  <input type = "checkbox name = "vehicles"
         value = "bike" /> Bike
</form>
… …

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

- The DOM 0 event model is somewhat limited in scope, but it is the only model supported by all browsers that support JavaScript.
- The DOM 2 model is supported by the Firefox version 2 (but not Internet Explorer, not even version 7).

Basic Concepts of Event Handling

- An event is a notification that something specific has happened, such as when a document finishes loading, a button is pushed or contents of a textbox is changed.
- An event handler is a script that is implicitly executed in response to an event happening.
- Event-driven programming is when parts of the programming are executed in an unpredictable sequence in response to specific events.

Events in JavaScript

- Events are objects in JavaScript with case-sensitive names, all of which are lower-case.
- Events are created by activities associated with specific elements of XHTML, e.g., clicking a radio button.
- The process of connecting an event handler to an event is called ***registration***. Registration can be done by:
  - Assigning tag attributes
  - Assigning handler addresses to object properties.
- Write should not be used in an event handler because it is asynchronous and thus what you write in the XHTML document is unpredictable.

Events, Attributes and Tags

- The same attribute can appear in several different tags.
- An XHTML element is said to get focus when the user puts the mouse over it and left-clicks or tabs over to the element.
- An element gets blurred when the user move the cursor away and left-clicks or when (s)he tabs away.

Registering an Event Handler

- There are a few ways of registering an event handler in DOM 0:
  - Assigning an event handler script to an event tag attribute:
    ```
    <input type = "button" id = "myButton"
           onclick = "alert(You clicked my button!');" />
    ```
  - Assigning an event to a function:
    ```
    <input type = "button" id = "myButton"
           onclick = "myButtonHandler();" />
    ```
  - Assigning to the associate event property on the button object:
```
document.getElementById("myButton").onclick
                                     = myButtonHandler;
```

Events, Attributes and Tags

Event	**Tag Attribute**
`blur`	`onblur`
`change`	`onchange`
`click`	`onclick`
`dblclick`	`ondblclcik`
`focus`	`onfocus`
`keydown`	`onkeydown`
`keypress`	`onkeypress`
`keyup`	`onkeyup`
`load`	`onload`

Events, Attributes and Tags (continue)

Event	**Tag Attribute**
`mousedown`	`onmousedown`
`mousemove`	`onmousemove`
`mouseout`	`onmouseout`
`mouseover`	`onmouseover`
`mouseup`	`onmouseup`
`reset`	`onreset`
`select`	`onselect`
`submit`	`onsubmit`
`unload`	`onunload`

Event Attributes and Their Tags

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 & loses input focus
	`<textarea>`	The text area is changed & loses input focus
	`<select>`	The selection element is changed & loses input focus
`onclick`	`<a>`	The user clicks on the link
	`<input>`	The input element is clicked

Attribute	Tag	Description
`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>`	The text area receives the input focus
	`<select>`	The selection element receives input focus
`onkeydown`	`<body>`, form elements	A key is pressed down
`onkeypress`	`<body>`, form elements	A key is pressed down & 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

Attribute	**Tag**	**Description**
`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>`	The mouse cursor is moved over the element
	`<textarea>`	The text area is selected within the text area
`onsubmit`	`<form>`	The submit button is pressed
`onunload`	`<body>`	The user exits the document

Handling Events from Body Elements

- The events most often created by body elements are **`load`** and **`unload`**.
- The **`unload`** even is probably more useful than the **`load`** event; we can use it to do some cleanup before a document is unloaded.

load.html

```
<!DOCTYPE html>
<!-- load.html
     An example to illustrate the load event
     A document for load.js
     -->
<html  lang = "en">
  <head>
    <title> onLoad event handler </title>
    <meta charset = "utf-8" />
    <script type = "text/javascript"
                   src = "load.js" >
    </script>
  </head>
  <body onload = "loadGreeting();">
    <p />
  </body>
</html>
```

load.js

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


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

Handling Events from Button Elements

- Buttons provide a simple and effective way to get input in a web document.
- The most commonly used event created by button actions is **`click`**.

Plain Buttons

- A plain button represents a simple situation.
- Let's assume that we create the button:

```
<input type = "button"
       name = "freeOffer"
       id = "freeButton">
```

- We can register a handler for this button:

```
<input type = "button" name = "freeButton" id =
"freeButton" onclick = "freeButtonHandler();" />
```

- Or we can register it by writing:

```
document.getElementById("freeButton").onclick
                                 = freeButtonHandler();
```

Radio Buttons

- Radio buttons handle a choice made from a set of options.
- In the following example, the calls to the event handlers send the value of the pressed radio button to the handler.

`radioclick.html`

```
<!DOCTYPE html>
<!-- radioClick.html
     An example of the use of the click event with
     radio buttons, registering the event handler by
     assignment to the button attributes
     -->
<html lang = "en">
  <head>
    <title> Illustrate messages for radio buttons
    </title>
    <meta charset = "utf-8">
    <script type = "text/javascript" src =
  "radio_click.js">
    </script>
  </head>
```

```
<body>
  <h4> Cessna single-engine airplane
                     descriptions </h4>
  <form id = "myForm" action ="handler">
    <p>
   <input type = "radio" name = "planeButton"
     value = "152" onclick = "planeChoice(152)" />
      Model 152
   <br />
   <input type = "radio" name = "planeButton"
     value = "172" onclick = "planeChoice(172)" />
      Model 172 (Skyhawk)
   <br />

   <input type = "radio" name = "planeButton"
    value = "182"  onclick = "planeChoice(182)" />
      Model 182 (Skylane)
      <br />
```

```
   <input type = "radio" name = "planeButton"
    value = "210"  onclick = "planeChoice(210)" />
      Model 210 (Centurian)
      <br />
    </p>
  </form>
</body>

</html>
```

radio_click.js

```
//  radio_click.js
//  An example of the use of the click event with
// radio buttons attributes

// The event handler for a radio button collection
function planeChoice(plane)  {
// Produce an alert message about the chosen
airplane
  switch(plane)  {
    case 152:
      alert("A small two-place airplane for"
                +  " flight training");
      break;
```

```
    case 172:
      alert("The smaller of two four-place "
              + " airplanes");
      break;
    case 182:
      alert("The larger of two four-place"
                  + " airplanes");
      break;
    case 210:
      alert("A six-place high-performance"
               + " airplane");
      break;
    default:
      alert("Error in JavaScript function"
              + " planeChoice");
      break;
  }
}
```

Registering an Event Hamdler by Assignment

- An event handler can be registered by assigning the name of the handler to the event properties of the radio button objects.
- Example:

```
document.getElementById
    ("myForm").elements[0].onclick
                             = planeChoice;
```

- This statement must follow both the handler function and the XHTML form specifications so that JavaScript has been able to see both.

radioclick2.html

```
<!-- radioClick2.html
     A document for radio_clicks2.js
     An example of the use of the click event with
  radio buttons,
     registering the event handler by assignment to
  the button
     attributes
     -->
<html lang = "en">
  <head>
    <title> Illustrate messages for radio buttons
  </title>
    <meta charset = "utf-8">
    <script type = "text/javascript"
            src = "radio_clicks2.js">
    </script>
  </head>
```

```
  <body>
    <h4> Cessna single-engine airplane
                      descriptions </h4>
    <form id = "myForm" action = "">
      <p>
        <label> <input type = "radio"
                 name = "planeButton"
                 value = "152"  />
          Model 152 </label>
        <br />
        <label> <input type = "radio"
                 name = "planeButton"
                 value = "172"  />
          Model 172 (Skyhawk) </label>
        <br />
        <label> <input type = "radio"
                 name = "planeButton"
                 value = "182"  />
          Model 182 (Skylane) </label>
        <br />
        <label> <input type = "radio"
                 name = "planeButton"
                 value = "210" />
          Model 210 (Centurian) </label>
        <br />
      </p>
    </form>
    <script type = "text/javascript"
            src = "radio_clicks2r.js">
    </script>
  </body>

</html>
```

radio_clicks2.js

```
// radioClicks2.html
// An example of the use of the click event with
// radio buttons, registering the event handler by
// assignment to the button attributes

// The event handler for a radio button
// collection
function planeChoice(plane)  {
  // Put the DOM address of the elements array
  // in a local variable
  var dom = document.getElementById("myForm");
```

```
// Determine which button was pressed
for (var index = 0;
      index < dom.planeButton.length;
                                 index++)  {
  if (dom.planeButton[index].checked)  {
    plane = dom.planeButton[index].value;
    break;
  }
}
// Produce an alert message about the chosen
// airplane
switch(plane)  {
  case "152":
    alert("A small two-place airplane for"
             + " flight training");
    break;
```

```
  case "172":
    alert("The smaller of two four-place "
               + "airplanes");
    break;

  case "182":
    alert("The larger of two"
           +" four-place airplanes");
    break;
  case "210":
    alert("A six-place high-performance "
                  + " airplane");
    break;
    default:
      alert("Error in JavaScript function"
               + " planeChoice");
      break;
  }
}
```

radio_clicks2r.js

```
//radio_clicks2r.js
// the event registering code for radio_clicks2
var dom = document.getElementById("myForm");
dom.elements[0].onclick = planeChoice;
dom.elements[1].onclick = planeChoice;
dom.elements[2].onclick = planeChoice;
dom.elements[3].onclick = planeChoice;
```

radio_clicks2r.js

Handling Events from Text Box and Password Elements

- Text boxes and password boxes can create four distinct events:
  - **`blur`**
  - **`focus`**
  - **`change`**
  - **`select`**

The Focus Event

- Focus occurs when a mouse hover over a given element and the mouse pointer is clicked.
- Blurring occurs when this focus is removed.
- This can be useful if there are fields on the form that should not be changed.
- Example:
  - A order pay where the client precalculates the price and we wish not to allow the user to alter the price being charged.

nochange.html

```
<!DOCTYPE html>

<!-- nochange.html
     This document illustates using the focus event
     to prevent the user from changing a text field
     -->

<html lang = "en">
  <head>
    <title> The focus event </title>
    <meta charset = "utf-8" />
    <script type = "text/javascript"
            src ="nochange.js">
    </script>
  </head>
```

```
  <body>
    <form action = "">
      <h3> Coffee Order Form </h3>

      <! -- A bordered table for item orders -->
          <table border = "border" >

      <! -- First the column headings -->
            <tr>
              <th> Product Name </th>
              <th> Price </th>
              <th> Quantity </th>
            </tr>

      <! -- Now, the table data entries -->
            <tr>
              <th> French Vanilla (1 lb.) </th>
              <td> $3.49 </td>
```

```
        <td> <input type = "text"
               id = "french"
               size = "2" /> </td>
      </tr>

      <tr>
        <th> Hazelnut Cream (1 lb.) </th>
        <td> $3.95 </td>
        <td> <input type = "text"
               id = "hazelnut"
               size = "2" /> </td>
      </tr>

      <tr>
        <th> Columbian (1 lb.) </th>
        <td> $4.59 </td>
        <td> <input type = "text"
               id = "columbian"
               size = "2" /> </td>
      </tr>
```

```
    </table>

  <!-- button for precomputation of
                     the total cost -->
  <p>
    <input type = "button"
     value = "Total Cost"
     onclick = "computeCost();" />

    <input type = "text"   size = "5"
     id = "cost"   onfocus = "this.blur();" />
  </p>
  <!-- The submit and reset buttons -->
  <p>
    <input type = "submit"
     value = "Submit Order" />

    <input type = "reset"
     value = "Clear Order Form" />
  </p>
```

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

nochange.js

```
//  This script illustrates using the focus event
//  to prevent the user from changing a text field

//  The event handler to compute the cost
function computeCost()  {
  var french
       = document.getElementById("french").value;
  var hazelnut
       = document.getElementById("hazelnut").value;
  var columbian
       = document.getElementById("columbian").value;
  // Compute the cost
  document.getElementById("cost").value =
  totalCost = french * 3.49 + hazelnut * 3.95
                  + columbian * 4.59;
}
```

Validating Form Input

- Validating form input on the client side using JavaScript results in a faster-responding server and less Internet traffic.
- If incorrect data is entered, the JavaScript function should:
  - produce an alert message about the erroneous entry.
  - put the relevant input element in focus. This is done by writing:
    `document.getElementById("phone").focus();`
  - select the element, highlighting it. This is done by writing:
    `document.getElementById("phone").select();`

Handling Invalid Input

- The handler should return false to tell the browser not to perform any default actions. This prevents bad data from being sent to the server.
- Checking passwords involve making sure that the original entry is made and that the second entry matches it. It should be called if the second entry blurs or if the submit button is pressed.

`pswdCheck.html`

```
<!DOCTYPE html>
<!-- pswdChk.html
     An example of input password checking, using
     the submit event
     -->
<html lang = "en">
  <head>
    <title> Illustrate password checking </title>
    <meta charset = "utf-8" />
    <script type = "text/javascript"
            src = "pswdchk.js">
    </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 = "second"
              size = "10" />
       </label>
       <br /> <br />

       <input type = "reset" name = "reset" />
       <input type = "submit" name = "submit" />
     </p>
   </form>
```

```
   <script type = "text/javascript"
           src = "pswdchkr.js">
   // Set submit button property to the event
   // handler

     document.getElementById("second").onblur
              = chkPasswords;
     document.getElementById("myForm").onsubmit
              = chkPasswords;
   </script>
  </body>

</html>
```

Checking the Forms of a Name and Telephone Number

- A name should be in a standard format; in this case it's *LastName*, *FirstName*, *MiddleInitial*. The pattern
  `/^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?$/`
  ensures that it is a last name beginning with a capital letter, a comma and optional space, a first name beginning with a capital letter, a comma and optional space, a capital letter and an optional period.
- The anchors `^` and `$` ensure that there is no other text in the box.
- Similarly, the phone number is checked with the pattern
  `/^d{3}-\d{3}-\d{4}$/`
  which ensure that there are 3, 3, and 4 digits separated by dashed with nothing else in the box.

`validator.html`

```
<!DOCTYPE html>

<!-- validator.html
     An example of input validation using the change
     and submit events
     -->

<html lang = "en">
  <head>
    <title> Illustrate form input validation
  </title>
    <meta charset = "utf-8">
    <script type = "text/javascript" src =
  "validator.js">
    </script>
  </head>
```

```
<body>
  <h3> Customer Information </h3>
  <form action = "">
    <p>
      <label>
        <input type = "text" id = "custName"
             onchange = "chkName();" />
        Name (last name, first name,
              middle initial)
      </label>
      <br /> <br />

      <label>
        <input type = "text" id = "phone" />
        Phone number (ddd-ddd-dddd)
      </label>
      <br /> <br />
```

```
      <input type = "reset" id = "reset" />
      <input type = "submit" id = "submit" />
    </p>
  </form>

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

  </script>
</body>

</html>
```

validator.js

```
// validator.js
//  An example of input validation using the change
//  and submit events.


// The event handler for the name text box
function chkName()  {
  var myName = document.getElementById("custName");

  // Test the format of the input name
  // Allow the spaces after the commas to be
  optional
  // Allow the period after the initial to be
  optional

  var pos = myName.value.search(
           /^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-
  Z]\.?$/);
```

```
  if (pos !=0)  {
    alert("The name you entered (" + myName.value +
          ") is not in the correct form. \n" +
          "The correct form is: " +
          "last-name, first-name, middle \n" +
          "Please go back and fix your name");
      myName.focus();
      myName.select();
      return false;
  }
  else
    return true;
}
```

```
// The event handler function for the phone number
// text box
function chkPhone()  {
  var myPhone = document.getElementById("phone");

  // Test the format of the input phone number
  var pos = myPhone.value.search
                     (/^\d{3}-\d{3}-\d{4}$/);

  if (pos != 0)  {
    alert("The phone number you enter ("
          + myPhone.value
          + ") is not in the correct form. \n"
          + "The correct form is: ddd-ddd-dddd \n"
          + "Please go back and fix your phone "
          + "number");
    myPhone.focus();
    myPhone.select();
    return false;
```

```
}
  else
    return true;
}
```

validatorr.js

```
// validatorr.js
// Register the event handlers for validator.html

// Set form element object properties to their
// corresponding event handler functions

document.getElementById("custName").onchange
                                        = chkName;

document.getElementById("phone").onchange = chkPhone;
```

The DOM 2 Event model

- The DOM 2 event model does not include the features of the DOM 1 event model. But without IE supporting it, there has not yet been a rush to move to it.
- The DOM 2 model is a model with a modular interface.
- One of its modules is Events, which has several submodules including **`HTMLEvents`** and **`MouseEvents`**.

`HTMLEvents` and `MouseEvents`

Module	Event Interface	Event Types
`HTMLEvents`	`Event`	`abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload`
`MouseEvents`	`MouseEvent`	`click, mousedown, mousemove, mouseout, mouseover, mouseup`

Event Propagation

- An event object is create at a node in the document tree; the node is called the ***target node***.
- Event creation causes a three-phase process:
  - Capturing phase – The even starts at the document's root node and propagates to the target node. Any registered handlers are checked to determine if they are enabled. Any enabled handler is executed.
  - The registered handler for the event is executed.
  - Bubbling phase – as the event bubbles back up the tree, any registered handlers are executed.
- Any handler can be stopped from further propagation using the **`stopPropagation()`** method.

Event Handler Registration

- Event handler registration is handled in DOM 2 model by the **`AddEventListener`** method.
- The call:

```
document.custName.addEventListener("change",
                                   chkName, false);
```

is of the text element *name of the event* *handler function*

is the handler enabled during the capturing phase?

An Example of the DOM 2 Event Model – **`validator2.html`**

```
<!DOCTYPE html>
<!-- validator2.html
     A document for validator2.js
     Creates text boxes for a name and a phone
     number
     Note: This document does nto work in IE6
     An example of input validation using the change
     and submit events
     -->

<html lang = "en">
  <head>
    <title> Illustrate form input validation with
           DOM 2 </title>
    <meta charset = "utf-8" />
    <!-- Script to define the event handlers -->
    <script type = "text/javascript"
            src = "validator2.js">
    </script>
  </head>

  <body>
    <h3> Customer Information </h3>
    <form action = "">
      <p>
        <label>
        <input type = "text" id = "custName" />
        Name (last name, first name, middle initial)
        </label>
        <br /> <br />
```

```
        <label>
        <input type = "text" id = "phone" />
        Phone number (ddd-ddd-dddd)
        </label>
        <br /> <br />

        <input type = "reset" />
        <input type = "submit"
               id = "submitButton" />
      </p>
    </form>

    <!-- Script for registering event handlers -->
    <script type = "text/javascript" src =
  "validator2r.js">
    </script>

  </body>
</html>
```

validator2.js

```
// validator2.js

//  An example of input validation using the change
  and
//  submit events, using the DOM 2 event model
//  NOte: This document does not work with IE6

//**********************************************//
// The event handler for the name text box
function chkName(event)  {

  // Get the target node of the event
  var myName = event.currentTarget;

  // Test the format of the input name
  // Allow the spaces after the commas to be optional
  // Allow the period after the initial to be
  // optional
```

```
  var pos = myName.value.search(
                      /^[A-Z][a-z]+, ?[A-Z][a-z]+,
  ?[A-Z]\.?$/);

  if (pos !=0)  {
    alert("The name you entered (" + myName.value +
          ") is not in the correct form. \n" +
          "The correct form is: " +
          "last-name, first-name, middle \n" +
          "Please go back and fix your name");
      // myName.focus();
      // myName.select();
  }
}
```

```
// The event handler function for the phone number
// text box
function chkPhone(event)  {
  var myPhone = event.currentTarget;

  // Test the format of the input phone number
  var pos = myPhone.value.search
                   (/^\d{3}-\d{3}-\d{4}$/);

  if (pos != 0)  {
    alert("The phone number you enter ("
         + myPhone.value +
          ") is not in the correct form. \n"
         + "The correct form is: ddd-ddd-dddd \n"
         + "Please go back and fix your phone "
         + "number");
  }
}
```

validator2r.js

```
// validator2r.js
//  The last part of validator2.  Registers the
//  event handlers
//  Note: This script does not work with IE6

//  Get the DOM addresses of the elements and
  register
//  the event handlers
    var customerNode
            = document.getElementById("custName");
    var phoneNode
            = document.getElementById("phone");
    customerNode.addEventListener("change",
                              chkName, false);
    phoneNode.addEventListener("change",
                              chkPhone, false);
```

The **`navigator`** Object

- The navigator object allows the script to determine the viewing browser and to call the appropriate handler for that browser

The **`canvas`** Element

- The **`canvas`** element crates a rectangle into which can be drawn bit-mapped graphics using JavaScript.
- The **`canvas`** element has three optional attributes:
  - **`height`** (non-negative integer, default = 150)
  - **`width`** (non-negative integer, default = 300)
  - **`id`**

The **canvas** Element – An Example

```
<canvas id = "myCanvas" height = "200"
                              width = "400">
Your browser does not support the canvas element
</canvas>
```

- The text in the element is display if the browser does not support the element.

navigate.html

```
<!DOCTYPE html>
<!-- navigator.html
     A document for navigator.js
     Call the event handler on load
     -->
<html lang = "en">
  <head>
    <title> navigate.html </title>
    <meta charset = "utf-8" />
    <!-- Script to define the event handlers -->
    <script type = "text/javascript"
            src = "navigate.js">
    </script>
  </head>
  <body onload = "navProperties()">
  </body>
</html>
```

navigate.js

```
// navigate.js
//  An example of using the navigator object

//  The event handler function to display the
//  browser name and its version number
function navProperties()  {
  alert("The browser is: " + navigator.appName
        + "\n" + "The version number is :" +
        navigator.appVersion + "\n");
}
```