Course: Web Programming Course Code: 15CS52T

Study Material

Note:

1. These notes are intended for use by students in 15CS52T.

2. These notes are provided free of charge and may not be sold in any shape or form.

3. These notes are not a substitute for material covered during course lectures.

4. If you miss a class, you should definitely obtain both these notes and notes written by a student who attended the class.

5. Material from these notes is obtained from various sources, including, but not limited to, the following:

- Programming the World Wide Web, 7th edition, Robert W. Sebesta
- http://www.tutorialspoint.com/
- http://www.w3schools.com/
- Web Programming – Building Internet Applications, 3rd edition, Chris Bates, Wiley Publisher

Unit-II JavaScript and XHTML documents and Dynamic documents with JavaScript

Session-2

Element access in JavaScript:

- Elements in XHTML have corresponding objects in JavaScript.
- The address for these objects is used by the event handling and by the code to make dynamic changes to documents.

There are three methods of accessing XHTML document elements in JavaScript

1. **By using Elements Array**
2. **By using Element Names**
3. **By Element IDs**

```
<html xmlns=”http://www.w4c.org/1999/xhtml”>
<head><title>Accessing the form elements</title>
</head>
 <body>
        <form name=”myform” action=” “>
          <input type=”button” name=”turniton”/>
 </body>
</html>
```

1. **By using Elements Array**
   - The address of JavaScript object that is associated with an XHTML element is its DOM address. The DOM address of the text field in above example using forms and elements arrays, is as follows:

     var dom=document.forms[0].elements[0]
   - The problem with this approach is that the address of elements depends on position of elements in the form that could change; if new elements

are added or existing elements are deleted i.e. the arrays forms and elements change.

var dom=document.forms[0].elements[1]

2. By Element Names

- The elements can be accessed by their names. This requires the element and all of its ancestors (except body) to have name attributes.

```
<html xmlns="http://www.w3c.org/1999/xhtml">
<head><title>Accessing the form elements</title>
</head>
<body>
<form name="myform" action=" ">
<input type="button" name="turniton"/>
</body>
</html>
```

- Using name attribute, the DOM address of button is:

var dom= document.myform.turniton

- One minor drawback of this approach is that the XHTML 1.1 standard does not allow the name attribute in the form clement, even though the attribute is legal for form elements.

3. By Element IDs

- The elements can be given unique identifier using id attribute. The element can be accessed by using **getElementById** method defined in DOM 1.
- As an element's id is unique in the document, this approach works, regardless of how deeply the element is nested in other elements in the document.

```
<html xmlns="http://www.w4c.org/1999/xhtml">
<head><title>Accessing the form elements</title>
```

```
</head>
  <body>
      <form name=”myform” action=” “>
        <input type=”button” id=”turniton”/>
  </body>
</html>
```

- The DOM address of button element is:

  var dom=document.getElementById(“turniton”)

- Buttons in a group of checkboxes often share the same name. The buttons in a radio button group always have the same name. In these cases, the names of the individual buttons obviously cannot be used in their DOM addresses. To access the arrays, the DOM address of the form object must first be obtained, as shown:

```
<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>
```

- The checked property of a checkbox object is set to true if the button is checked. For the preceding sample checkbox group, the following code would count the number of checkboxes that were checked:

```
var numChecked = 0;
var dom = document.getElementById("vehicleGroup");
for (index = 0; index < dom.vehicles.length; index++)
  if (dom.vehicles[index].checked)
    numChecked++;
```

- Radio buttons can be addressed and handled exactly as are the checkboxes in the foregoing code.

Events and Event Handling

- A complete and comprehensive event model was specified by DOM 2. The DOM 2 model is supported by the FX2 browser.
- Events and event handling is divided into two parts:
  - ✓ One for the DOM 0 model and one for the DOM 2 model**.**

Basic Concepts of Event Handling

- JavaScript detects certain activities of the browser and the user and provide computation when those activities occur.
- These computations are specified with a special form of programming called **event driven programming.**
- 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.
- 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

- HTML 4 defined a **collection of events** that browsers implement and with which JavaScript can deal.
- These **events** are associated with **XHTML tag attributes,** which can be used to **connect the events to handlers**.
- Table lists the most commonly used events and their associated tag attributes.

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

Table 2.1: Events and their tag attributes

In many cases, the same attribute can appear in several different tags. The Table 2 . 2 shows (1st column) the most commonly used attributes related to events, (column 2) tags that can include the attributes, and (column 3) the circumstances under which the associated events are created.

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

Table 2.2: Event attributes and their tags

The two ways to register an event handler in the DOM 0 event model are:

1. **By assigning the event handler script to a tag attribute.**
   - 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 following example of a button element:

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

2. **By assigning the handler addresses to object properties.**
   - An event handler function could also be registered by assigning its name to the associated event property on the button object, as shown in the following example:

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

   - This statement must follow both the handler function and the form element so that JavaScript sees both before assigning the property.
   - Notice that only the name of the handler function is assigned to the property—it is neither a string nor a call to the function.