PEARSON NEW INTERNATIONAL EDITION

Programming the World Wide Web

Robert W. Sebesta

Seventh Edition

PEARSON

Pearson New International Edition

Programming the World Wide Web
Robert W. Sebesta
Seventh Edition

PEARSON

which the buttons appear. To access the arrays, the DOM address of the form object must first be obtained, as in the following example:

```
<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 implicit array, `vehicles`, has three elements, which reference the three objects associated with the three checkbox elements in the group. This array provides a convenient way to search the list of checkboxes in a group. 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 counts 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.

4 Events and Event Handling

The HTML 4.0 standard provided the first specification of an event model for markup documents. This model is sometimes referred to as the DOM 0 event model. Although the DOM 0 event model is limited in scope, it is the only event model supported by all browsers that support JavaScript. A complete and comprehensive event model was specified by DOM 2. The DOM 2 model is supported by the FX3 and Chrome browsers. However, although IE9 supports it, earlier versions of IE do not. Our discussion of events and event handling is divided into two parts, one for the DOM 0 model and one for the DOM 2 model.

4.1 Basic Concepts of 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 in the browser, such as the completion of the loading of a document, or a browser user action, such as a mouse click on a form button. Strictly speaking, an event is an object that is implicitly created by the browser and the JavaScript system in response to something having happened.

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.

If you are familiar with the exceptions and exception-handling capabilities of a programming language such as C++ or Java, you should see the close relationship between events and exceptions. Events and exceptions occur at unpredictable times, and both often require some special program actions.

Because events are JavaScript objects, their names are case sensitive. The names of all event objects have only lowercase letters. For example, `click` is an event, but `Click` is not.

Events are created by activities associated with specific HTML elements. For example, the `click` event can be caused by the browser user clicking a radio button or the link of an anchor tag, among other things. Thus, an event's name is only part of the information pertinent to handling the event. In most cases, the specific HTML element that caused the event is also needed.

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. These are further discussed in Sections 5 and 6.

The `write` method of `document` should never be used in an event handler. Remember that a document is displayed as its markup is parsed by the browser. Events usually occur after the whole document is displayed. If `write` appears in an event handler, the content produced by it might be placed over the top of the currently displayed document.

The remainder of this section and Sections 5 through 7 describe the DOM 0 event model and some of its uses.

4.2 Events, Attributes, and Tags

HTML4 defined a collection of events that browsers implement and with which JavaScript can deal. These events are associated with HTML tag attributes, which can be used to connect the events to handlers. The attributes have names that are closely related to their associated events. Table 1 lists the most commonly used events and their associated tag attributes.

Table 1 Events and their tag attributes

Events	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`

In many cases, the same attribute can appear in several different tags. The circumstances under which an event is created are related to a tag and an attribute, and they can be different for the same attribute when it appears in different tags.

An HTML element is said to *get focus* when the user puts the mouse cursor over it and clicks the left mouse button. An element can also get focus when the user tabs to the element. When a text element has focus, any keyboard input goes into that element. Obviously, only one text element can have focus at one time. An element becomes blurred when the user moves the cursor away from the element and clicks the left mouse button or when the user tabs away from the element. An element obviously becomes blurred when another element gets focus. Several nontext elements can also have focus, but the condition is less useful in those cases.

Table 2 shows (1) the most commonly used attributes related to events, (2) tags that can include the attributes, and (3) the circumstances under which the associated events are created. Only a few of the situations shown in the table are discussed in this chapter.

Table 2 Event attributes and their tags

Attributes	Tag	Description
`onblur`	`<a>`	The link loses focus.
	`<button>`	The button loses focus.
	`<input>`	The input element loses focus.
	`<textarea>`	The text area loses focus.
	`<select>`	The selection element loses focus.
`onchange`	`<input>`	The input element is changed and loses focus.
	`<textarea>`	The text area is changed and loses focus.
	`<select>`	The selection element is changed and loses focus.
`onclick`	`<a>`	The user clicks the link.
	`<input>`	The input element is clicked.
`ondblclick`	Most elements	The user double-clicks the left mouse button.
`onfocus`	`<a>`	The link acquires focus.
	`<input>`	The input element acquires focus.
	`<textarea>`	A text area acquires focus.
	`<select>`	A selection element acquires focus.
`onkeydown`	`<body>`, form elements	A key is pressed.
`onkeypress`	`<body>`, form elements	A key is pressed 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.

Table 2 Event attributes and their tags (*continued*)

Attributes	Tag	Description
`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 following 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;
```

This statement must follow both the handler function and the form element so that JavaScript has seen 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.

5 Handling Events from Body Elements

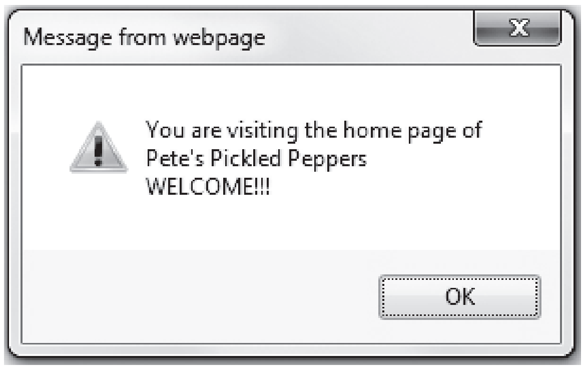
The events most often created by body elements are `load` and `unload`. As our first example of event handling, we consider the simple case of producing an alert message when the body of the document has been loaded. In this case, we use the `onload` attribute of `<body>` to specify the event handler:

```
<!DOCTYPE html>
<!-- load.html
     A document for load.js
     -->
<html lang = "en">
  <head>
    <title> load.html </title>
    <meta charset = "utf-8" />
    <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!!!");
}
```

Figure 3 shows a browser display of `load.html`.



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

Figure 3 Display of `load.html`