

Working with Dynamic Content and Styles

Objectives


- Learn how to create dynamic content under the IE DOM
- Understand the methods and properties of nodes and the node tree
- Learn to create element and text nodes
- Understand how to attach nodes to a Web page document

2

*New Perspectives on
JavaScript and AJAX, 2nd Edition*

- 2

New Perspectives on
JavaScript and AJAX, 2nd Edition




Objectives

- Apply node properties and styles to create dynamic content
- Work with the properties and methods of attribute nodes
- Work with element attributes

3

*New Perspectives on
JavaScript and AJAX, 2nd Edition*

- 3 *New Perspectives on JavaScript and AJAX, 2nd Edition*



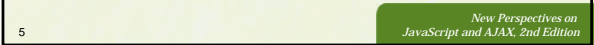
Objectives

- Hide and redisplay Web page objects
- Understand how to create recursive functions to navigate a node tree
- Learn to work with the properties and methods of style sheet objects

4

*New Perspectives on
JavaScript and AJAX, 2nd Edition*

- 4 *New Perspectives on JavaScript and AJAX, 2nd Edition*

[illegible]

Introducing Dynamic Content

- Inserting HTML Content into an Element
 - Generating a table of contents involves working with **dynamic content**, which is content determined by the operation of a script running within the browser
 - One property that can be used to write content in an element is the innerHTML property
`object.innerHTML = content`

6

New Perspectives on
JavaScript and AJAX, 2nd Edition


- 6

New Perspectives on
JavaScript and AJAX, 2nd Edition

Introducing Dynamic Content

- Inserting HTML Content into an Element

```
function makeTOC() {
    var TOC = document.getElementById("toc");
    TOC.innerHTML = "<h1>Table of Contents</h1>";
}
```



The screenshot shows a web page with a heading 'Table of Contents' and a subheading 'The Constitution of the United States'. A red box labeled 'new h1 heading' points to the 'Table of Contents' heading.

New Perspectives on
JavaScript and AJAX, 2nd Edition

Introducing Dynamic Content

- Dynamic Content in Internet Explorer
 - The innerHTML property is not part of the official specifications for the W3C document object model
 - However, since it has proven valuable and easy to use, it is supported by all browsers
 - If you want to change both the content and the HTML element itself, you use the outerHTML property
`object.outerHTML = content;`

New Perspectives on
JavaScript and AJAX, 2nd Edition

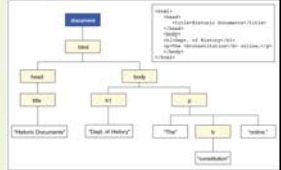
Working with Nodes

- A **node** represents an object within the Web page and Web browser
- The text within an HTML tag can also be treated as a node. For example, the tag
`<h1>Table of Contents</h1>`
consists of two nodes: one node for the h1 element and one node for the text string, Table of Contents, contained within that element

New Perspectives on
JavaScript and AJAX, 2nd Edition

Working with Nodes

- The Node Tree
 - Nodes are arranged into a hierarchal structure called a **node tree**, which indicates the relationship between each of the nodes



The diagram shows a node tree with a root node 'document' which has children 'html' and 'body'. 'html' has children 'head' and 'body'. 'head' has children 'meta' and 'title'. 'body' has children 'h1' and 'p'. 'h1' has children 'Table of Contents' and 'Table of Contents'.

New Perspectives on
JavaScript and AJAX, 2nd Edition

Working with Nodes

- The Node Tree
 - The parent of all nodes within a document is the **root node**

Expression	Description
<code>node.firstChild</code>	Returns the first child of <i>node</i>
<code>node.lastChild</code>	Returns the last child of <i>node</i>
<code>node.childNodes</code>	Returns a collection containing the children of <i>node</i>
<code>node.previousSibling</code>	Returns the sibling prior to <i>node</i>
<code>node.nextSibling</code>	Returns the sibling after <i>node</i>
<code>node.ownerDocument</code>	Returns the root node of the document
<code>node.parentNode</code>	Returns the parent of <i>node</i>

New Perspectives on
JavaScript and AJAX, 2nd Edition

Working with Nodes

- Node types, names, and values

Node	.nodeType	.nodeName	.nodeValue
Element	1	ELEMENT NAME	null
Attribute	2	attribute name	attribute value
Text	3	#text	text string
Comment	8	#comment	comment text
Document	9	#document	null

New Perspectives on
JavaScript and AJAX, 2nd Edition

Working with Nodes

- Node types, names, and values

Node	nodeType	nodeName	nodeValue
Document	9	#document	null
html	1	HTML	null
head	1	HEAD	null
body	1	BODY	null
title	1	TITLE	null
"Historic Documents"	3	#text	Historic Documents
h1	1	H1	null
"Dept. of History"	3	#text	Dept. of History
p	1	P	null
"The"	3	#text	The
h	1	H	null
"constitution"	3	#text	constitution
"online"	3	#text	online

New Perspectives on
JavaScript and AJAX, 2nd Edition

13

Working with Nodes

- Creating and Attaching Nodes

Method	Description
<code>document.createAttribute(attr)</code>	Creates an attribute node with the name <i>attr</i> .
<code>document.createComment(text)</code>	Creates a comment node containing the comment text string <i>text</i> .
<code>document.createElement(elem)</code>	Creates an element node with the name <i>elem</i> .
<code>document.createTextNode(text)</code>	Creates a text node containing the text string <i>text</i> .
<code>node.cloneNode(deep)</code>	Creates a copy of <i>node</i> . If the Boolean parameter <i>deep</i> is true, the copy extends to all descendants of the node object; otherwise, only <i>node</i> is copied.

New Perspectives on
JavaScript and AJAX, 2nd Edition

14

Working with Nodes

- Creating and Attaching Nodes
 - Unattached nodes and node trees are known as **document fragments** and exist only in a browser's memory

Method	Description
<code>node.appendChild(new)</code>	Appends a new child node to <i>node</i> , attaching it as the last child node.
<code>node.insertBefore(new, child)</code>	Inserts a new child node into <i>node</i> , placing it before the child node; if no child is specified the new child node is added as the last child node.
<code>node.normalize()</code>	Traverses all child nodes of <i>node</i> ; any adjacent text nodes are merged into a single text node.
<code>node.removeChild(old)</code>	Removes the child node <i>old</i> from <i>node</i> .
<code>node.replaceChild(new, old)</code>	Replaces the child node <i>old</i> with the child node <i>new</i> .

New Perspectives on
JavaScript and AJAX, 2nd Edition

15

Working with Nodes

- Creating and Attaching Nodes

Code	Node Tree
<pre>newP = document.createElement("P"); newI = document.createElement("I"); text1 = document.createTextNode(" Documents"); text2 = document.createTextNode(" Historic");</pre>	
<pre>newP.appendChild(text1); newI.appendChild(text2);</pre>	
<pre>newP.insertBefore(newI, text1);</pre>	
<p>Final HTML fragment:</p> <pre><p><i>Historic</i></p> Documents</p></pre>	

New Perspectives on
JavaScript and AJAX, 2nd Edition

16

Creating a List of Heading Elements

- Looping Through the Child Node Collection

```
function makeTOC() {
    var TOC = document.getElementsByTagName("toc");
    var innerHTML = "<div><table></table>";
    var TOCList = document.createElement("div");
    TOC.appendChild(TOCList);

    function createList(object, list) {
        for (var n = object.firstChild; n != null; n = n.nextSibling) {
            // loop through all of the nodes within object
        }
    }
}
```

creates the list
items for the
table of contents

New Perspectives on
JavaScript and AJAX, 2nd Edition

17

Creating a List of Heading Elements

- Matching the Heading Elements

```
function createList(object, list) {
    for (var n = object.firstChild; n != null; n = n.nextSibling) {
        // loop through all of the nodes within object

        var nodeLevel = levelNum(n);
        if (nodeLevel != -1) {
            // node represents a section heading
        }
    }
}
```

locates the section
headings within
the object

New Perspectives on
JavaScript and AJAX, 2nd Edition

18

Creating a List of Heading Elements

- Creating the List Item Elements

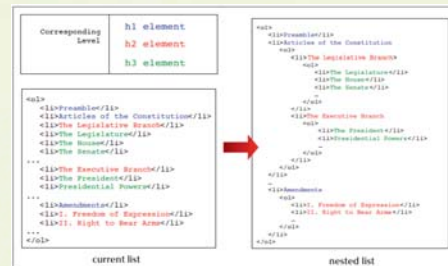
```
function createlist(object, list) {
  for (var n = object.firstChild; n != null; n = n.nextSibling) {
    // Loop through all of the nodes within object
    var nodeLevel = levelNum(n);
    if (nodeLevel != -1) {
      // node represents a section heading
      // create a list item to match
      var listitem = document.createElement("li");
      listitem.innerHTML = n.innerHTML;
      list.appendChild(listitem);
    }
  }
}
```

text of the list item
comes from the text
of the section heading

19

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List



20

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List

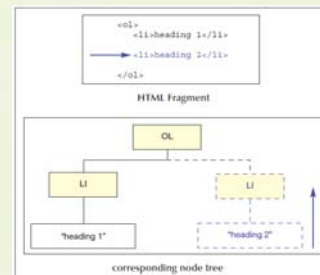
```
function createlist(object, list) {
  var prevLevel = 0; // level of the previous TOC entry
  for (var n = object.firstChild; n != null; n = n.nextSibling) {
    // Loop through all of the nodes within object
    var nodeLevel = levelNum(n);
    if (nodeLevel != -1) {
      // node represents a section heading
      // create a list item to match
      var listitem = document.createElement("li");
      listitem.innerHTML = n.innerHTML;
      if (nodeLevel == prevLevel) {
        // append the entry to the current list
      }
      else if (nodeLevel > prevLevel) {
        // append the entry to a new nested list
      }
      else if (nodeLevel < prevLevel) {
        // append the entry to a higher-level list
      }
    }
  }
}
```

replaces the statement
to append the list item

21

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List



22

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List

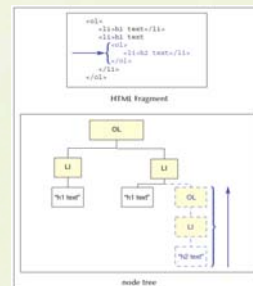
```
function createlist(object, list) {
  var prevLevel = 0; // level of the previous TOC entry
  for (var n = object.firstChild; n != null; n = n.nextSibling) {
    // Loop through all of the nodes within object
    var nodeLevel = levelNum(n);
    if (nodeLevel != -1) {
      // node represents a section heading
      // create a list item to match
      var listitem = document.createElement("li");
      listitem.innerHTML = n.innerHTML;
      if (nodeLevel == prevLevel) {
        // append the entry to the current list
      }
      else if (nodeLevel > prevLevel) {
        // append the entry to a new nested list
      }
      else if (nodeLevel < prevLevel) {
        // append the entry to a higher-level list
      }
    }
  }
}
```

replaces the statement
to append the list item

23

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List



24

New Perspectives on
JavaScript and AJAX, 2nd Edition

Creating a Nested List

```

<ol>
  <li>OK 1st</li>
  <li>OK 2nd</li>
  <ol>
    <li>OK 3rd</li>
    <li>OK 4th</li>
    <li>OK 5th</li>
  </ol>
  <li>OK 6th</li>
</ol>

```

HTML elements

node tree

- OK
 - 11
 - 11
 - OK
 - 11
 - 11
 - OK
 - 11
 - 11
 - 11
 - 11

Working with Attributes

```

<ul>
  <li>a href = "#head1">Preamble</li>
  <li>a href = "#head1">Articles of the Constitution</li>
  <ul>
    <li>a href="#head1">Legislative Branch</li>
    <li>a href = "#head1">Section 1: The Legislature</li>
    <li>a href = "#head1">Section 2: The House</li>
    <li>a href = "#head1">Section 3: The Senate</li>
  </ul>
</ul>

```

Table of contents with links

```

<h1 id = "head1">Preamble</h1>
<h1 id = "head2">Articles of the Constitution</h1>
<h2 id = "head1">Legislative Branch</h2>
<h3 id = "head1">Section 1: The Legislature</h3>
<h3 id = "head1">Section 2: The House</h3>
<h3 id = "head1">Section 3: The Senate</h3>

```

section headings with id values

New Perspectives on
JavaScript and AJAX, 2nd Edition

26

Working with Attributes

- Attribute Nodes

The diagram illustrates the concept of attribute nodes in a DOM tree. It is divided into two main sections: "HTML fragment" and "node tree".

HTML fragment: Contains the following code:

```
<li>
  <a href = "#head4">
    The House
  </a>
</li>
```

node tree: Shows a vertical hierarchy of nodes represented by boxes:


- The top node is labeled "li".
- Below it is a node labeled "A".
- Below "A" is a node labeled "The House".

A dashed red oval labeled `href = "#head4"` is connected to the "A" node, representing the attribute node.

27

New Perspectives on
JavaScript and AJAX, 2nd Edition

- 27



Working with Attributes


- Attribute Nodes

Method	Description
<code>document.createAttribute(atts)</code>	Creates an attribute node with the name <i>atts</i>
<code>node.getAttribute(atts)</code>	Returns the value of an attribute <i>atts</i> from a <i>node</i> to which it has been attached
<code>node.hasAttribute(atts)</code>	Returns a Boolean value indicating whether <i>node</i> has the attribute <i>atts</i>
<code>node.removeAttribute(atts)</code>	Removes the attribute <i>atts</i> from <i>node</i>
<code>node.removeAttributeNode(atts)</code>	Removes an attribute node <i>atts</i> from <i>node</i>
<code>node.setAttribute(atts, value)</code>	Creates or changes the value of the attribute <i>atts</i> of <i>node</i>

New Perspectives on
JavaScript and AJAX, 2nd Edition

28

- 28



Working with Attributes

- Attributes as Object Properties
 - The document object model also supports a shorthand way of applying attributes as properties of an object


```
object.attr = value;
```
 - To test whether the listItem node has an id attribute, you can use the following expression:


```
listItem.hasAttribute("id")
```

29

New Perspectives on
 JavaScript and AJAX, 2nd Edition

- 29

Working with Attributes

- Setting the Section Heading IDs

```
function createList(object, list) {  
    var prevLevel = 0; // level of the previous toc entry  
    var headNum = 0; // running count of section headings  
    for (var n = object.firstChild; n !== null; n = n.nextSibling) {  
        // loop through all of the nodes within object  
        var nextLevel = 1 + prevLevel(n);  
        if (nextLevel != 2) {  
            // node represents a section heading  
            // insert id for the section heading if necessary  
            headNum++;  
            if (n.id == "" || {n.id == "head" : headNum});  
            // create a list item to hold it  
            var listItem = document.createElement("li");  
            listItem.innerHTML = n.innerHTML;  
        }  
    }  
}
```

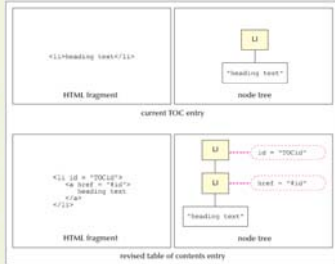
adds IDs to section headings that don't already have one

New Perspectives on
JavaScript and AJAX, 2nd Edition

- 30

Working with Attributes

Inserting Links



31

New Perspectives on
JavaScript and AJAX, 2nd Edition

Working with Attributes

Inserting Links

```
var nodeLevel = levelNum(n);
if (nodeLevel != -1) {
    // node represents a section heading
    // insert id for the section heading if necessary
    headNum++;
    if (n.id == "") {n.id = "head" + headNum;}

    // create a list item to match
    var listItem = document.createElement("li");
    listItem.id = "li" + n.id;

    // create a hypertext link to the section heading
    var linkItem = document.createElement("a");
    linkItem.innerHTML = n.innerHTML;
    linkItem.href = "p" + n.id;

    // Append the hypertext link to the list entry
    listItem.appendChild(linkItem);

    if (nodeLevel == prevLevel) {
        // append the entry to the current list
        list.appendChild(listItem);
    }
}
```

delete the line to place the heading content into the list item

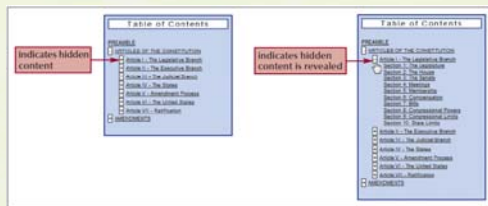
creates the hypertext link

appends the link to the list item

32

New Perspectives on
JavaScript and AJAX, 2nd Edition

Expanding and Collapsing a Document



33

New Perspectives on
JavaScript and AJAX, 2nd Edition

Expanding and Collapsing a Document

Creating a plus/minus Box

```
<li id="TOChead34">
  <span--></span>
  <a href="#head34">
    Amendments
  </a>
  <ol>
    <li id="TOChead35">
      <a href="#head35">
        I. Freedom of Expression
      </a>
    </li>
    <li id="TOChead36">
      <a href="#head36">
        II. Right to Bear Arms
      </a>
    </li>
  </ol>
</li>
```

plus/minus box

plus/minus box

HTML fragment

Rendered elements

34

New Perspectives on
JavaScript and AJAX, 2nd Edition

Expanding and Collapsing a Document

Creating a plus/minus Box

```
else if (nodeLevel > prevLevel) {
    // append the entry to a new nested list
    var nestedList = document.createElement("ul");
    nestedList.appendChild(listItem);

    list.appendChild(nestedList);

    // add plus/minus box before the text of the nested list
    var plusMinusBox = document.createElement("span");
    plusMinusBox.innerHTML = "+";
    nestedList.parentNode.insertBefore(plusMinusBox, nestedList);

    list = nestedList;
    prevLevel = nodeLevel;
}
```

places a plus/minus box before each nested list

35

New Perspectives on
JavaScript and AJAX, 2nd Edition

Expanding and Collapsing a Document

Adding Event Handlers to the plus/minus Boxes

```
else if (nodeLevel > prevLevel) {
    // append the entry to a new nested list
    var nestedList = document.createElement("ul");
    nestedList.appendChild(listItem);

    list.appendChild(nestedList);

    // add plus/minus box before the text of the nested list
    var plusMinusBox = document.createElement("span");
    plusMinusBox.innerHTML = "+";
    nestedList.parentNode.insertBefore(plusMinusBox, nestedList);

    list = nestedList;
    prevLevel = nodeLevel;
}

// add event handler to the plus/minus box
function expandCollapse() {
    var plusMinusBox = document.getElementById("plusMinusBox");
    plusMinusBox.addEventListener("click", expandCollapse, false);
    plusMinusBox.innerHTML = "-";
    plusMinusBox.parentNode.removeChild(plusMinusBox);
    plusMinusBox.parentNode.appendChild(plusMinusBox);
    plusMinusBox.innerHTML = "+";
}
```

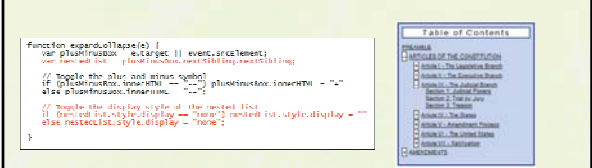
runs the expandCollapse function when the user clicks a plus/minus box

36

New Perspectives on
JavaScript and AJAX, 2nd Edition

Expanding and Collapsing a Document

- Hiding and Display Objects



Expanding and Collapsing a Document

- Expanding and Collapsing the Source Document

```

function expandCollapse(e) {
    $plugin.nu = e.target ? e.target.nextElementSibling :
    var nestedList = $plugin.nu.nextElementSibling;

    // toggle the plus and minus symbol
    if ($plugin.nu.innerHTML == "+" ) $plugin.nu.innerHTML = "-"
    else $plugin.nu.innerHTML = "+"

    // toggle the display style of the nested list
    if (nestedList.style.display == "none") nestedList.style.display = ""
    else nestedList.style.display = "none"
}

function expandCollapseDoc() {
    var displayStatus = ""
    for (var n = $plugin.doc.firstChild; n != null; n = n.nextSibling) {
        var nodeLevel = findLevel(n)
        // determine the display status of the toc entry
        if (nodeLevel != -1) {
            if (n.nodeType == 1) { // node represents a page element
                // apply the current display status to the node
            }
        }
    }
}

```

Expanding and Collapsing a Document

- Expanding and Collapsing the Source Document

```

function expandCollapsable() {
  var collCollapsible = $(".collapsible");
  var currentColl = $(".collapsible:visible:first");
  if (currentColl.length) {
    // toggle the current visible collapsible
    currentColl.toggle("slide", {
      duration: 200,
      easing: "swing",
      complete: function() {
        // expand and collapse the source document to match the TOC
        expandCollapsable();
      }
    });
  }
}

function expandCollapsable() {
  var collCollapsible = $(".collapsible");
  var currentColl = $(".collapsible:visible:first");
  if (currentColl.length) {
    // toggle the current visible collapsible
    currentColl.toggle("slide", {
      duration: 200,
      easing: "swing",
      complete: function() {
        // expand and collapse the source document to match the TOC
        expandCollapsable();
      }
    });
  }
}

function expandCollapsable() {
  var collCollapsible = $(".collapsible");
  var currentColl = $(".collapsible:visible:first");
  if (currentColl.length) {
    // toggle the current visible collapsible
    currentColl.toggle("slide", {
      duration: 200,
      easing: "swing",
      complete: function() {
        // expand and collapse the source document to match the TOC
        expandCollapsable();
      }
    });
  }
}

```

[illegible]

```
function exportAllLevels() {
    var displayStatus = " ";
    for (var i = 0; i < tree.children.length; i++) {
        var nodeLevel = levelFromId(i);
        if (nodeLevel != 1) {
            // determine the display status of the tree entry
            var TCEEntry = document.getElementById("TCE" + i + 1);
            if (document.getElementById("displayStatus" + i) != null) {
                var displayStatus = "none";
            }
        }
    }
}
```

```


}
if (nodeType == 1) { // node represents a page element
    // apply the current display status to the node
    node.style.display = displayStatus;
}
}

```

applies the display status to the current node in the source document

Expanding and Collapsing a Document

- Testing the Dynamic TOC

[illegible]

Traversing the Node Tree with Recursion

- **Recursion** is a programming technique in which a function calls itself repeatedly until a stopping condition is met

```
function countNodes(node, nodeCount) {
    for (var n = node.firstChild; n != null; n = n.nextSibling) {
        nodeCount++;
        countNodes(n, nodeCount);
    }
    return nodeCount;
}
```

Switching Between Style Sheets



Switching Between Style Sheets

- Style sheets can be classified as persistent, preferred, and alternate:
 - Persistent style sheets** are always active
 - Preferred style sheets** are turned on by default, but can be turned off by actions of the user
 - Alternate style sheets** are not turned on by default, but can be turned on as an alternate to the preferred style sheet

43

New Perspectives on
JavaScript and AJAX, 2nd Edition

Switching Between Style Sheets



44

New Perspectives on
JavaScript and AJAX, 2nd Edition

Switching Between Style Sheets

- Code to populate the allStyles array

```
addEvent(window, "load", makeStyleButtons, false);
var allStyles = new Array();
function makeStyleButtons() {
    var allLinks = document.getElementsByTagName("link");
    // Create an array of preferred or alternate style sheets
    for (var i = 0; i < allLinks.length; i++) {
        if (allLinks[i].rel == "stylesheet" || allLinks[i].rel == "alternate stylesheet") {
            allStyles.push(allLinks[i]);
        }
    }
    // allStyles array contains all link elements created
    // for preferred or alternate style sheets
}
```

45

New Perspectives on
JavaScript and AJAX, 2nd Edition

Switching Between Style Sheets

- Creating a form button for each style sheet

```
// Create an array of preferred or alternate style sheets
for (var i = 0; i < allLinks.length; i++) {
    if (allLinks[i].rel == "stylesheet" || allLinks[i].rel == "alternate stylesheet") {
        allStyles.push(allLinks[i]);
    }
}
// Create buttons for each preferred or alternate style sheet
var styleBox = document.createElement("div");
for (var i = 0; i < allStyles.length; i++) {
    styleBox.innerHTML += document.createElement("input");
    styleBox.value = allStyles[i].title + " view";
    styleBox.setAttribute("type", "button");
    styleBox.setAttribute("id", "style-button-" + i);
    styleBox.appendChild(styleButton);
}
```

46

New Perspectives on
JavaScript and AJAX, 2nd Edition

Switching Between Style Sheets

- You can disable a style sheet using the command `styleSheet.disabled = true`
- You can enable it with the command `styleSheet.disabled = false`
- Code to initialize the style sheets

```
// Create buttons for each preferred or alternate style sheet
for (var i = 0; i < allStyles.length; i++) {
    // Initialize the style sheet
    if (allStyles[i].rel == "stylesheet") {
        allStyles[i].disabled = false;
    } else {
        allStyles[i].disabled = true;
    }
    styleButton = document.createElement("input");
    styleButton.type = "button";
    styleButton.value = allStyles[i].title + " view";
    styleButton.setAttribute("id", "style-button-" + i);
    styleBox.appendChild(styleButton);
}
```

47

New Perspectives on
JavaScript and AJAX, 2nd Edition

Switching Between Style Sheets

- Code to switch between styles

```
// Apply an event handler to the style button
styleButton.onclick = changeStyle;
styleBox.appendChild(styleButton);
// Define the styles of the box containing the buttons
styleBox.style.width = "120px";
styleBox.style.height = "30px";
styleBox.style.backgroundColor = "#f0f0f0";
styleBox.style.border = "1px solid #ccc";
styleBox.style.margin = "5px 5px 10px 10px";
// Add the style box to the source document
var sourceDoc = document.getElementsByTagName("div");
sourceDoc.insertBefore(styleBox, sourceDoc.firstChild);
function changeStyle() {
    for (var i = 0; i < allStyles.length; i++) {
        if (allStyles[i].title == this.value) {
            allStyles[i].disabled = false;
            allStyles[i].disabled = true;
        }
    }
}
```

48

New Perspectives on
JavaScript and AJAX, 2nd Edition



Switching Between Style Sheets

- Properties of the style sheet object

Property	Description
<code>stylesheet.cssText</code>	The text of the declarations in the style sheet (IE DOM)
<code>stylesheet.disabled</code>	Returns a Boolean value indicating whether the style sheet has been disabled (true) or has been enabled (false)
<code>stylesheet.href</code>	The url of the style sheet; for embedded style sheets, the href value is an empty text string [read-only]
<code>stylesheet.media</code>	A text string containing the list of media types associated with the style sheet [read-only]
<code>stylesheet.rules</code>	Returns the collection of rules within the style sheet (IE DOM)
<code>stylesheet.cssRules</code>	Returns the collection of rules within the style sheet (W3C DOM)
<code>stylesheet.title</code>	The title of the style sheet [read-only]
<code>stylesheet.type</code>	The MIME type of the style sheet [read-only]