Document Object Model (DOM): Objects and Collections
Chapter 12

OBJECTIVES

- In this chapter you will learn:
  - How to use JavaScript and the W3C Document Object Model to create dynamic web pages.
  - The concept of DOM nodes and DOM trees.
  - How to traverse, edit and modify elements in an XHTML document.
  - How to change CSS styles dynamically.
  - To create JavaScript animations.

Chapter 12 Sections

- 12.1 Introduction
- 12.2 Modeling a Document: DOM Nodes and Trees
- 12.3 Traversing and Modifying a DOM Tree
- 12.4 DOM Collections
- 12.5 Dynamic Styles
- 12.6 Summary of the DOM Objects and Collections
- 12.7 Wrap-Up
- 12.8 Web Resources
What is DOM?

- The World Wide Web Consortium (W3C) defines the Document Object Model (DOM)
- Platform- and language-neutral
- Permits script to access and update the content, structure, and style of a document

How Do Web Authors Use the DOM?

- To access everything in the web page document
  - Make numerous content updates
  - Work with content in separate document fragments
- Working together with the Dynamic HTML (DHTML) Object Model

DOM Advantages

- Enhances a Web author’s ability to build and manage complex documents and data
- Moving an object from one part of the document to another is easy
DOM and Dynamic Effects

- XHTML elements can be treated as objects with the DOM
- Attributes of XHTML elements can be treated as properties of those objects
- Thus, scripting can be used to address objects through their id attribute, and property values (attributes) can be changed at runtime with JavaScript
- Result = “dynamic” effects

DHTML and DOM

- Microsoft’s DHTML Object Model provides access to almost all document elements and to all attributes of an element
- W3C DOM is consistent with the DHTML Object Model in that every element and every attribute is accessible in script
- W3C DOM is an evolution from the DHTML Object Model
<?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">
<!-- Fig. 12.2: dom.html -->
<!-- Basic DOM functionality. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Basic DOM Functionality</title>
<style type="text/css">
h1, h3       { text-align: center;
font-family: tahoma, geneva, sans-serif }
p            { margin-left: 5%;
margin-right: 5%;
font-family: arial, helvetica, sans-serif }
ul           { margin-left: 10% }

a            { text-decoration: none }
a:hover      { text-decoration: underline }
.nav         { width: 100%;
border-top: 3px dashed blue;
padding-top: 10px }
.highlighted { background-color: yellow }
.submit      { width: 120px }
</style>
<script type="text/javascript">
var currentNode; // stores the currently highlighted node
var idcount = 0; // used to assign a unique id to new elements

// get and highlight an element by its id attribute
function byId() {
var id = document.getElementById("gbi").value;
var target = document.getElementById(id);
if (target)
switchTo(target);
}

// insert a paragraph element before the current element
// using the insertBefore method
function insert() {
var newNode = createNewNode(document.getElementById("ins").value);
diff(newNode.currentNode.parentNode.insertBefore(newNode,
currentNode));
switchTo(newNode);
}

// append a paragraph node as the child of the current node
function appendNode() {
var newNode = createNewNode(document.getElementById("append").value);
currentNode.appendChild(newNode);
switchTo(newNode);
}

// replace the currently selected node with a paragraph node
function replaceCurrent() {
var newNode = createNewNode(document.getElementById("replace").value);
currentNode.parentNode.replaceChild(newNode, currentNode);
switchTo(newNode);
}

// remove the current node
function remove() {
if (currentNode.parentNode == document.body)
alert("Can't remove a top-level element.");
else {
var oldNode = currentNode;
switchTo(oldNode.parentNode);
currentNode.removeChild(oldNode);
}
}

// get and highlight the parent of the current node
function parent() {
var target = currentNode.parentNode;
if (target != document.body)
switchTo(target);
else
alert("No parent.");
}
</script>
</head>
<body>
</body>
</html>
Fig. 12.2 | Basic DOM functionality (Part 4 of 14).

```javascript
// helper function that returns a new paragraph node containing
// a unique id and the given text
function createNewNode( text ) {
    var newNode = document.createElement( "p" );
    nodeId = "new" + idcount;
    ++idcount;
    newNode.id = nodeId;
    text = "[" + nodeId + "] " + text;
    newNode.appendChild(document.createTextNode( text ) );
    return newNode;
}
```

Fig. 12.2 | Basic DOM functionality (Part 5 of 14).

```html
<script>
// helper function that switches to a new currentNode
function switchTo( newNode ) {
    currentNode.className = "highlighted"; // highlight new node
    document.getElementById( "gbi" ).value = currentNode.id;
}
</script>
</body>
</html>

Fig. 12.2 | Basic DOM functionality (Part 6 of 14).

<p>The Document Object Model (DOM) allows for quick, dynamic access to all elements in an XHTML document for manipulation with JavaScript.</p>

<p>For more information, check out the "JavaScript and the DOM" section of Deitel's <a>JavaScript Resource Center.</a></p>

<ol>
    <li>getElementById and parentNode</li>
    <li>insertBefore and appendChild</li>
    <li>replaceChild and removeChild </li>
</ol>

```html
<form onsubmit="return false" action="">
<table>
    <tr>
        <td><input type="text" id="gbi" value="bigheading" /></td>
        <td><input type="submit" value="Get By id" onclick="byId()" class="submit" /></td>
    </tr>
    <tr>
        <td><input type="text" id="ins" /></td>
        <td><input type="submit" value="Insert Before" onclick="insert()" class="submit" /></td>
    </tr>
    <tr>
        <td><input type="text" id="append" /></td>
        <td><input type="submit" value="Append Child" onclick="appendNode()" class="submit" /></td>
    </tr>
    <tr>
        <td><input type="text" id="replace" /></td>
        <td><input type="submit" value="Replace Current" onclick="replaceCurrent()" class="submit" /></td>
    </tr>
    <tr><td> </td><td><input type="submit" value="Remove Current" onclick="remove()" class="submit" /></td>
    <tr>
    <tr><td> </td><td><input type="submit" value="Get Parent" onclick="parent()" class="submit" /></td>
</table>
</form>
```

6
<?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">
<!-- Fig. 12.3: collections.html -->
<!-- Using the links collection. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Using Links Collection</title>
<style type="text/css">
body          { font-family: arial, helvetica, sans-serif }
h1            { font-family: tahoma, geneva, sans-serif; text-align: center }
p             { margin: 5% }
p a           { color: #aa0000 }
.links        { font-size: 14px; text-align: justify; margin-left: 10%; margin-right: 10% }
.link a       { text-decoration: none }
.link a:hover { text-decoration: underline }
</style>
<script type="text/javascript">
<!--
function processlinks()
{
var linkslist = document.links; // get the document's links
var contents = "Links in this page:
<br />| 

// concatenate each link to contents
for ( var i = 0; i < linkslist.length; i++ )
"}
</script>
</head>
<body>

<!-- End of page -->
</body>
</html>
Fig. 12.3 | Using the links collection (Part 2 of 3).

```javascript
var currentLink = linkslist[i];
contents += '<span class = 'link'> + currentLink.innerHTML.link( currentLink.href ) + '</span>' | ;
```

Fig. 12.3 | Using the links collection (Part 3 of 3).

```html
<fig>
<Deitel Resource Centers>
</Deitel Resource Centers>
```
<?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">
<!-- Fig. 12.5: coverviewer.html -->
<!-- Dynamic styles used for animation. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
title>Deitel Book Cover Viewer</title>
<style type="text/css">
.thumbs { width: 192px; height: 370px; padding: 5px; float: left }
.mainimg { width: 289px; padding: 5px; float: left }
.imgCover { height: 373px }
img { border: 1px solid black }
</style>
<script type="text/javascript">
<!--
var interval = null; // keeps track of the interval
var speed = 6; // determines the speed of the animation
var count = 0; // size of the image during the animation

// called repeatedly to animate the book cover
function run()
{
count += speed;
// stop the animation when the image is large enough
if ( count >= 375 )
{
window.clearInterval( interval );
interval = null;
}
var bigImage = document.getElementById( "imgCover" );
bigImage.style.width = .7656 * count + "px";
bigImage.style.height = count + "px";
}
// inserts the proper image into the main image area and begins the animation
function display( imgfile )
{
if ( interval )
return;
var bigImage = document.getElementById( "imgCover" );
var newNode = document.createElement( "img" );
newNode.id = "imgCover";
newNode.src = "fullsize/" + imgfile;
newNode.alt = "Large image";
newNode.className = "imgCover";
newNode.style.width = "0px";
newNode.style.height = "0px";
bigImage.parentNode.replaceChild( newNode, bigImage );
count = 0; // start the image at size 0
-->
</script>
</head>
<body>

</body>
</html>
Fig. 12.5 | Dynamic styles used for animation (Part 3 of 7).

```javascript
interval = window.setInterval( "run()");
```
<table>
<thead>
<tr>
<th>Object or Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objects</strong></td>
<td></td>
</tr>
<tr>
<td>Window</td>
<td>Represents the browser window and provides access to the document object contained in the window. Also contains the history and location objects.</td>
</tr>
<tr>
<td>Document</td>
<td>Represents the XHTML document rendered in a window. The document object provides access to every element in the XHTML document and allows dynamic modification of the XHTML document. Contains several collections for accessing all elements of a given type.</td>
</tr>
<tr>
<td>Body</td>
<td>Provides access to the body element of an XHTML document.</td>
</tr>
<tr>
<td>History</td>
<td>Keeps track of the sites visited by the browser user. The object provides a script programmer with the ability to move forward and backward through the visited sites.</td>
</tr>
<tr>
<td>Location</td>
<td>Contains the URL of the rendered document. When this object is set to a new URL, the browser immediately navigates to the new location.</td>
</tr>
<tr>
<td><strong>Collections</strong></td>
<td></td>
</tr>
<tr>
<td>Anchors</td>
<td>Collection contains all the anchor elements (a) that have a name or id attribute. The elements appear in the collection in the order in which they were defined in the XHTML document.</td>
</tr>
<tr>
<td>Forms</td>
<td>Contains all the form elements in the XHTML document. The elements appear in the collection in the order in which they were defined in the XHTML document.</td>
</tr>
<tr>
<td>Images</td>
<td>Contains all the image elements in the XHTML document. The elements appear in the collection in the order in which they were defined in the XHTML document.</td>
</tr>
<tr>
<td>Links</td>
<td>Contains all the anchor elements (a) with an href property. The elements appear in the collection in the order in which they were defined in the XHTML document.</td>
</tr>
</tbody>
</table>