12

Document Object Model (DOM): Objects and Collections

Our children may learn about heroes of the past. Our task is to make ourselves architects of the future.

—Jomo Mzee Kenyatta

Though leaves are many, the root is one.

—William Butler Yeats

The thing that impresses me most about America is the way parents obey their children.

—Duke of Windsor

Most of us become parents long before we have stopped being children.

—Mignon McLaughlin

To write it, it took three months; to conceive it three minutes; to collect the data in it—all my life.

—F. Scott Fitzgerald

Sibling rivalry is inevitable. The only sure way to avoid it is to have one child.

—Nancy Samalin

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.

Outline

12.1 Introduction

- **The Document Object Model gives you access to all the elements on a web page. Using JavaScript, you can create, modify and remove elements in the page dynamically.**

Software Engineering Observation 12.1

With the DOM, XHTML elements can be treated as objects, and many attributes of XHTML elements can be treated as properties of those objects. Then, objects can be scripted (through their *id* attributes) with JavaScript to achieve dynamic effects.

12.2 Modeling a Document: DOM Nodes and Trees

- `getElementById` method
  - Returns objects called DOM nodes
  - Every element in an XHTML page is modeled in the web browser by a DOM node
- The nodes in a document make up the page's DOM tree, which describes the relationships among elements
- Nodes are related to each other through child-parent relationships
- A node may have multiple children, but only one parent
- Nodes with the same parent node are referred to as siblings
- Firefox's DOM Inspector and the IE Web Developer Toolbar allow you to see a visual representation of a document's DOM tree and information about each node
- The document node in a DOM tree is called the root node, because it has no parent

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 12.1: domtree.html -->
6  <!-- Demonstration of a document's DOM tree. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>DOM Tree Demonstration</title>
10    </head>
11    <body>
12       <h1>An XHTML Page</h1>
13       <p>This page contains some basic XHTML elements. We use the Firefox
14          DOM Inspector and the IE Developer Toolbar to view the DOM tree
15          of the document, which contains a DOM node for every element in
16          the document.</p>
17       <p>Here's a list:</p>
18       <ul>
19          <li>One</li>
20          <li>Two</li>
21          <li>Three</li>
22       </ul>
23    </body>
24 </html>
```

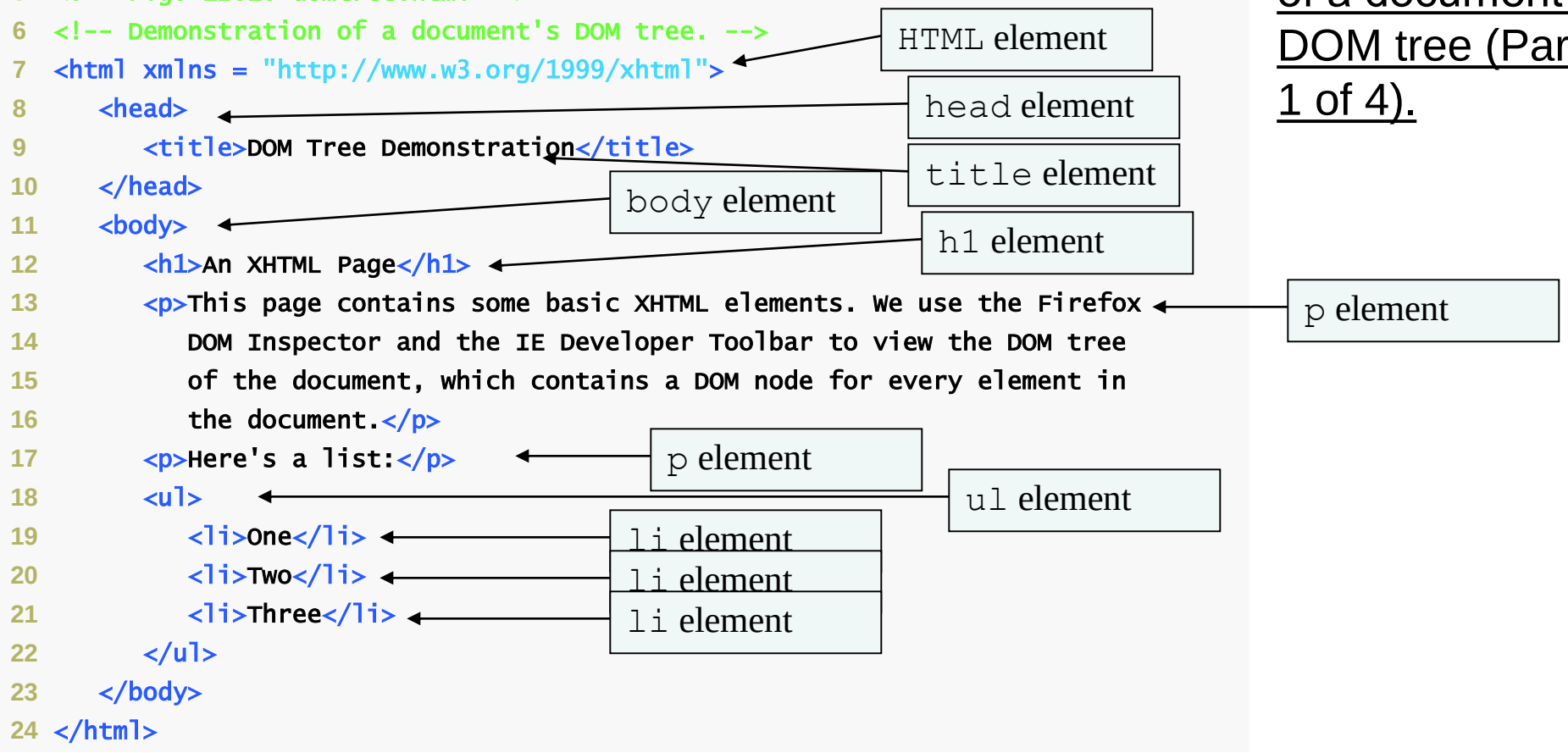

Fig. 12.1 | Demonstration of a document's DOM tree (Part 1 of 4).

a) The XHTML document is rendered in Firefox.

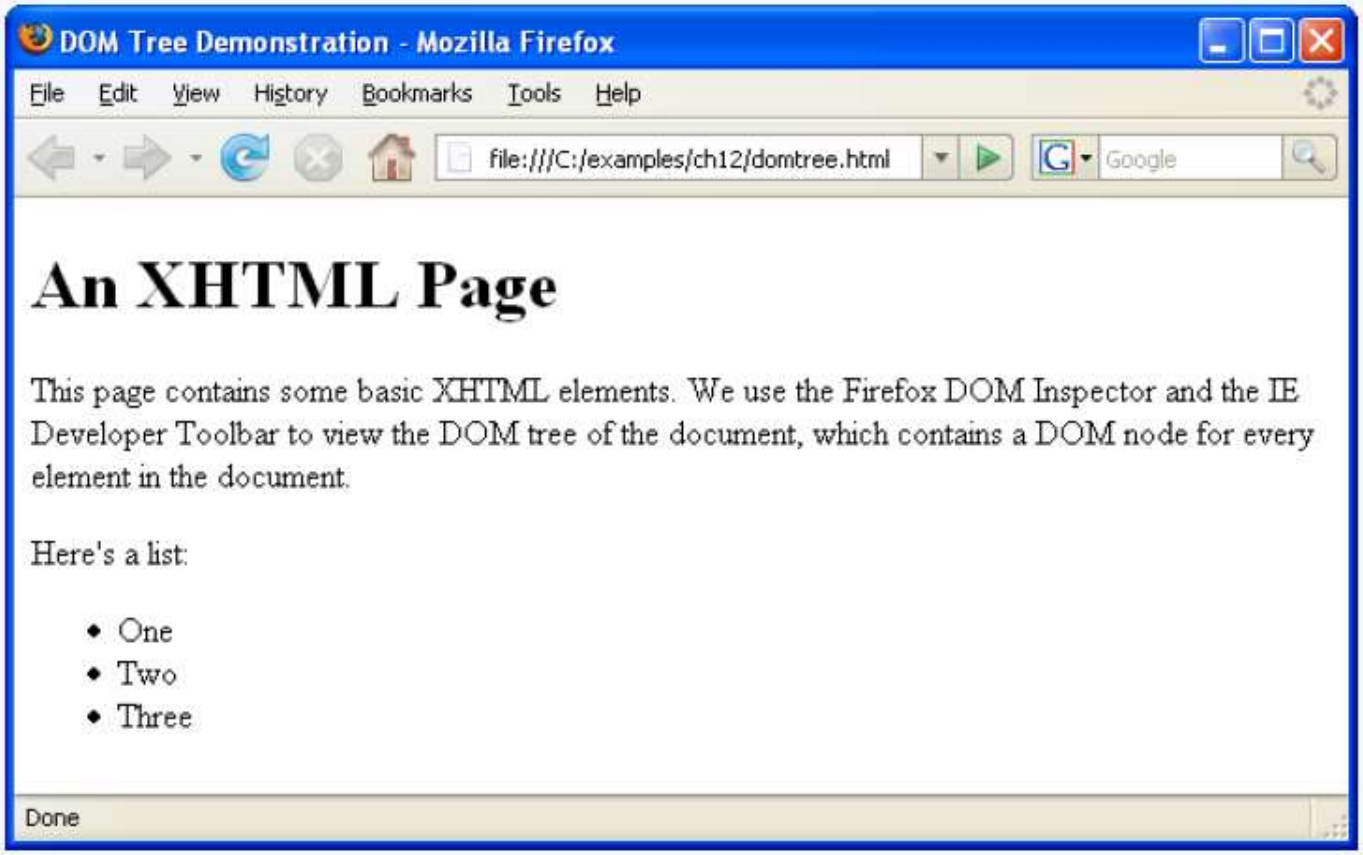

Fig. 12.1 | Demonstration of a document's DOM tree (Part 2 of 4).

b) The Firefox DOM inspector displays the document tree in the left panel. The right panel shows information about the currently selected node.

HEAD and BODY nodes are siblings

The BODY node is the parent of the H1 node

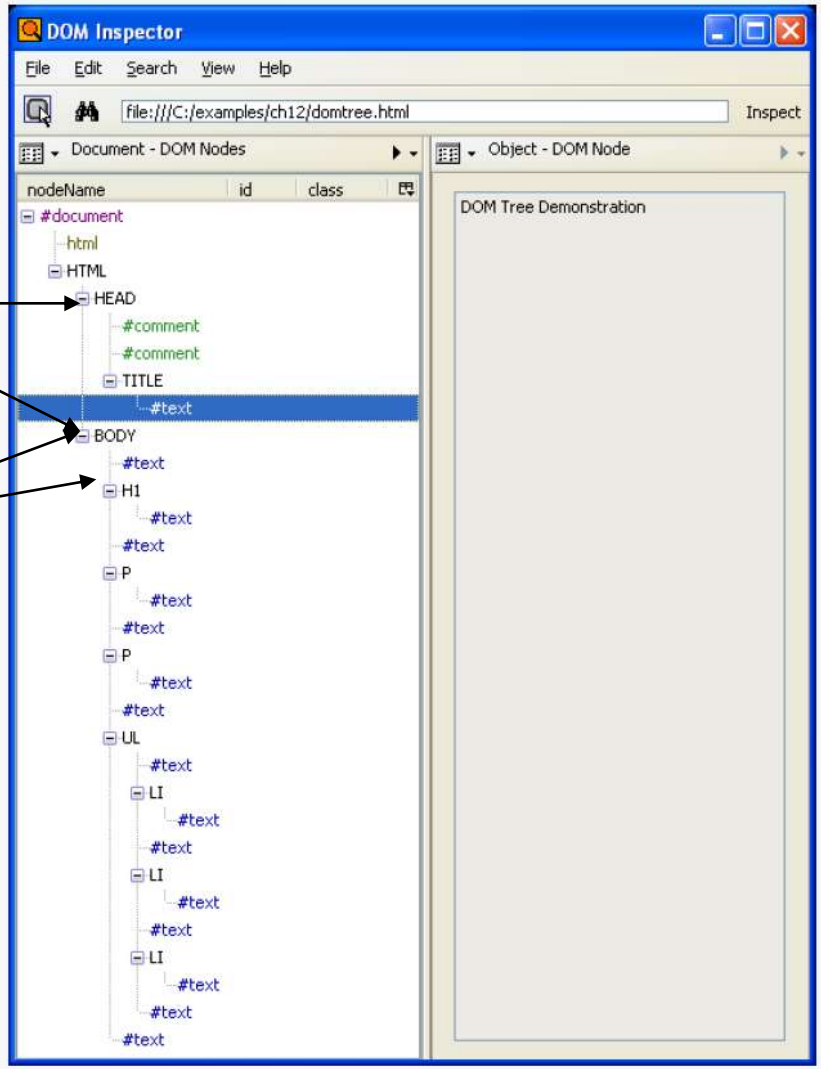

Fig. 12.1 | Demonstration of a document's DOM tree (Part 3 of 4).

c) The Internet Explorer Web Developer Toolbar displays much of the same information as the DOM inspector in Firefox in a panel at the bottom of the browser window.

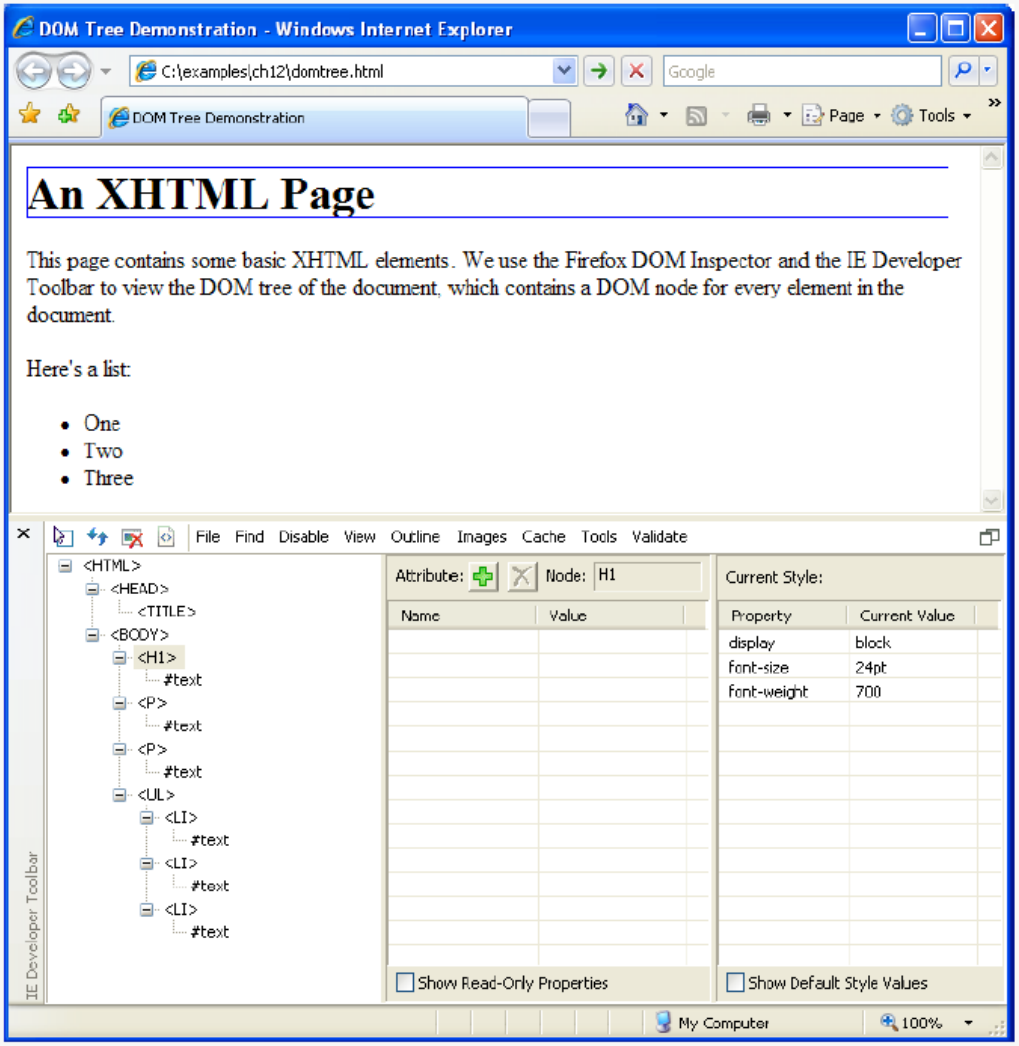

Fig. 12.1 | Demonstration of a document's DOM tree (Part 4 of 4).

12.3 Traversing and Modifying a DOM Tree

- The `className` property of a DOM node allows you to change an XHTML element's `class` attribute
- The `id` property of a DOM node controls an element's `id` attribute

12.3 Traversing and Modifying a DOM Tree (Cont.)

- `document` object `createElement` method
  - Creates a new DOM node, taking the tag name as an argument. Does not insert the element on the page.
- `document` object `createTextNode` method
  - Creates a DOM node that can contain only text. Given a string argument, `createTextNode` inserts the string into the text node.
- Method `appendChild`
  - Called on a parent node to insert a child node (passed as an argument) after any existing children
- `parentNode` property of any DOM node contains the node's parent
- `insertBefore` method
  - Called on a parent with a new child and an existing child as arguments. The new child is inserted as a child of the parent directly before the existing child.
- `replaceChild` method
  - Called on a parent, taking a new child and an existing child as arguments. The method inserts the new child into its list of children in place of the existing child.
- `removeChild` method
  - Called on a parent with a child to be removed as an argument.

Fig. 12.2 | Basic DOM functionality (Part 1 of 14).

```
<?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

```

Creates a class to highlight text

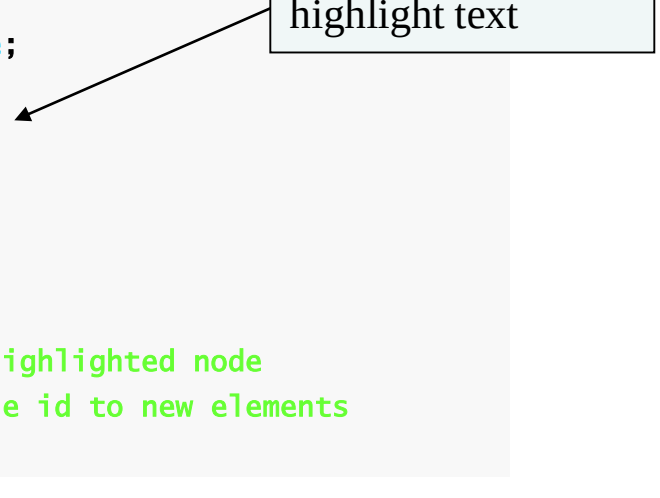

Fig. 12.2 | Basic DOM functionality (Part 2 of 14).

```
30    // get and highlight an element by its id attribute
31    function byId()
32    {
33       var id = document.getElementById( "gbi" ).value;
34       var target = document.getElementById( id );
35
36       if ( target )
37          switchTo( target );
38    } // end function byId
39
40    // insert a paragraph element before the current element
41    // using the insertBefore method
42    function insert()
43    {
44       var newNode = createNewNode(
45          document.getElementById( "ins" ).value );
46       currentNode.parentNode.insertBefore( newNode, currentNode );
47       switchTo( newNode );
48    } // end function insert
49
50    // append a paragraph node as the child of the current node
51    function appendNode()
52    {
53       var newNode = createNewNode(
54          document.getElementById( "append" ).value );
55       currentNode.appendChild( newNode );
56       switchTo( newNode );
57    } // end function appendNode
58
```

Calls function `switchTo` if the object can be found

Calls function `createNewNode` to make a new `p` node with the text in the `ins` text field

Inserts `newNode` as a child of the parent node, directly before `currentNode`

Highlights `newNode` with function `switchTo`

Inserts `newNode` as a child of the current node

Fig. 12.2 | Basic DOM functionality (Part 3 of 14).

```
59       // replace the currently selected node with a paragraph node
60       function replaceCurrent()
61       {
62          var newNode = createNewNode(
63             document.getElementById( "replace" ).value );
64          currentNode.parentNode.replaceChild( newNode, currentNode );
65          switchTo( newNode );
66       } // end function replaceCurrent
67
68       // remove the current node
69       function remove()
70       {
71          if ( currentNode.parentNode == document.body )
72             alert( "Can't remove a top-level element." );
73          else
74          {
75             var oldNode = currentNode;
76             switchTo( oldNode.parentNode );
77             currentNode.removeChild( oldNode );
78          }
79       } // end function remove
80
81       // get and highlight the parent of the current node
82       function parent()
83       {
84          var target = currentNode.parentNode;
85
86          if ( target != document.body )
87             switchTo( target );
88          else
89             alert( "No parent." );
90       } // end function parent
```

Gets the parent of `currentNode`, then inserts `newNode` into its list of children in place of `currentNode`

Ensures that top-level elements are not removed

Highlights `oldNode`'s parent

Removes `oldNode` from the document

Gets the parent node

Makes sure the parent is not the `body` element

Fig. 12.2 | Basic DOM functionality (Part 4 of 14).

```
91
92          // helper function that returns a new paragraph node containing
93          // a unique id and the given text
94          function createNewNode( text )
95          {
96             var newNode = document.createElement( "p" );
97             nodeId = "new" + idcount;
98             ++idcount;
99             newNode.id = nodeId;
100            text = "[" + nodeId + "] " + text;
101            newNode.appendChild(document.createTextNode( text ) );
102            return newNode;
103         } // end function createNewNode
104
105         // helper function that switches to a new currentNode
106         function switchTo( newNode )
107         {
108            currentNode.className = ""; // remove old highlighting
109            currentNode = newNode;
110            currentNode.className = "highlighted"; // highlight new node
111            document.getElementById( "gbi" ).value = currentNode.id;
112         } // end function switchTo
113         // -->
114      </script>
115   </head>
116   <body onload = "currentNode = document.getElementById( 'bigheading' )">
117      <h1 id = "bigheading" class = "highlighted">
118         [bigheading] DHTML Object Model</h1>
119      <h3 id = "smallheading">[smallheading] Element Functionality</h3>
```

Creates (but does not insert) a new `p` node

Creates a unique `id` for the new node

Creates new text node with the contents of variable `text`, then inserts this node as a child of `newNode`

Changes `class` attribute to unhighlight old node

Highlights `currentNode`

Assigns `currentNode`'s `id` to the input field's `value` property

```
<p id = "para1">[para1] The Document Object Model (DOM) allows for
   quick, dynamic access to all elements in an XHTML document for
   manipulation with JavaScript.</p>
<p id = "para2">[para2] For more information, check out the
   "JavaScript and the DOM" section of Deitel's
   <a id = "link" href = "http://www.deitel.com/javascript">
      [link] JavaScript Resource Center.</a></p>
<p id = "para3">[para3] The buttons below demonstrate:(list)</p>
<ul id = "list">
   <li id = "item1">[item1] getElementById and parentNode</li>
   <li id = "item2">[item2] insertBefore and appendChild</li>
   <li id = "item3">[item3] replaceChild and removeChild </li>
</ul>
<div id = "nav" class = "nav">
   <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>
```

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

```
150                <td><input type = "text" id = "replace" /></td>
151                <td><input type = "submit" value = "Replace Current"
152                   onclick = "replaceCurrent()" class = "submit" /></td>
153             </tr><tr><td />
154                <td><input type = "submit" value = "Remove Current"
155                   onclick = "remove()" class = "submit" /></td>
156             </tr><tr><td />
157                <td><input type = "submit" value = "Get Parent"
158                   onclick = "parent()" class = "submit" /></td>
159             </tr>
160          </table>
161       </form>
162    </div>
163 </body>
164</html>
```

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

a) This is the page when it first loads. It begins with the large heading highlighted.

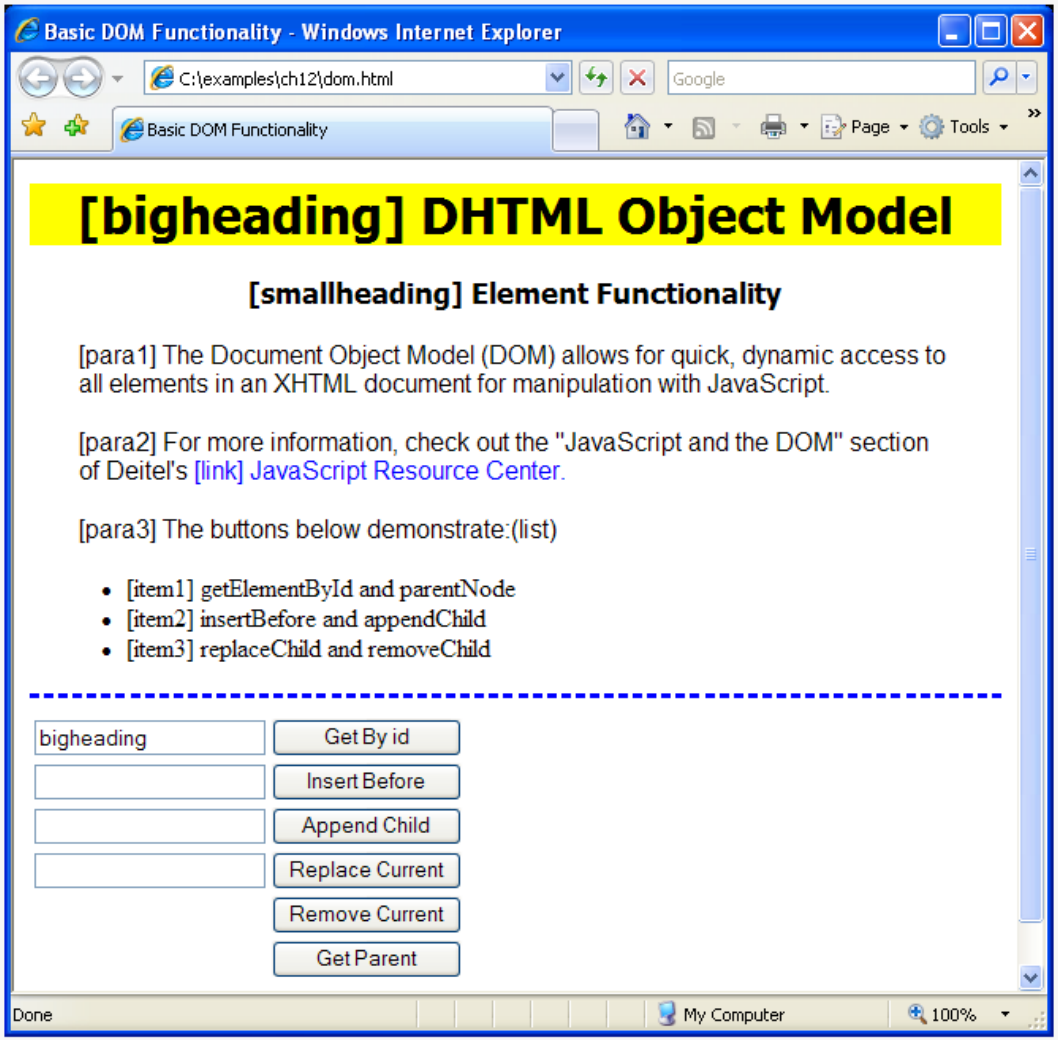

Fig. 12.2 | Basic DOM functionality (Part 7 of 14).

b) This is the document after using the Get By id button to select para3.

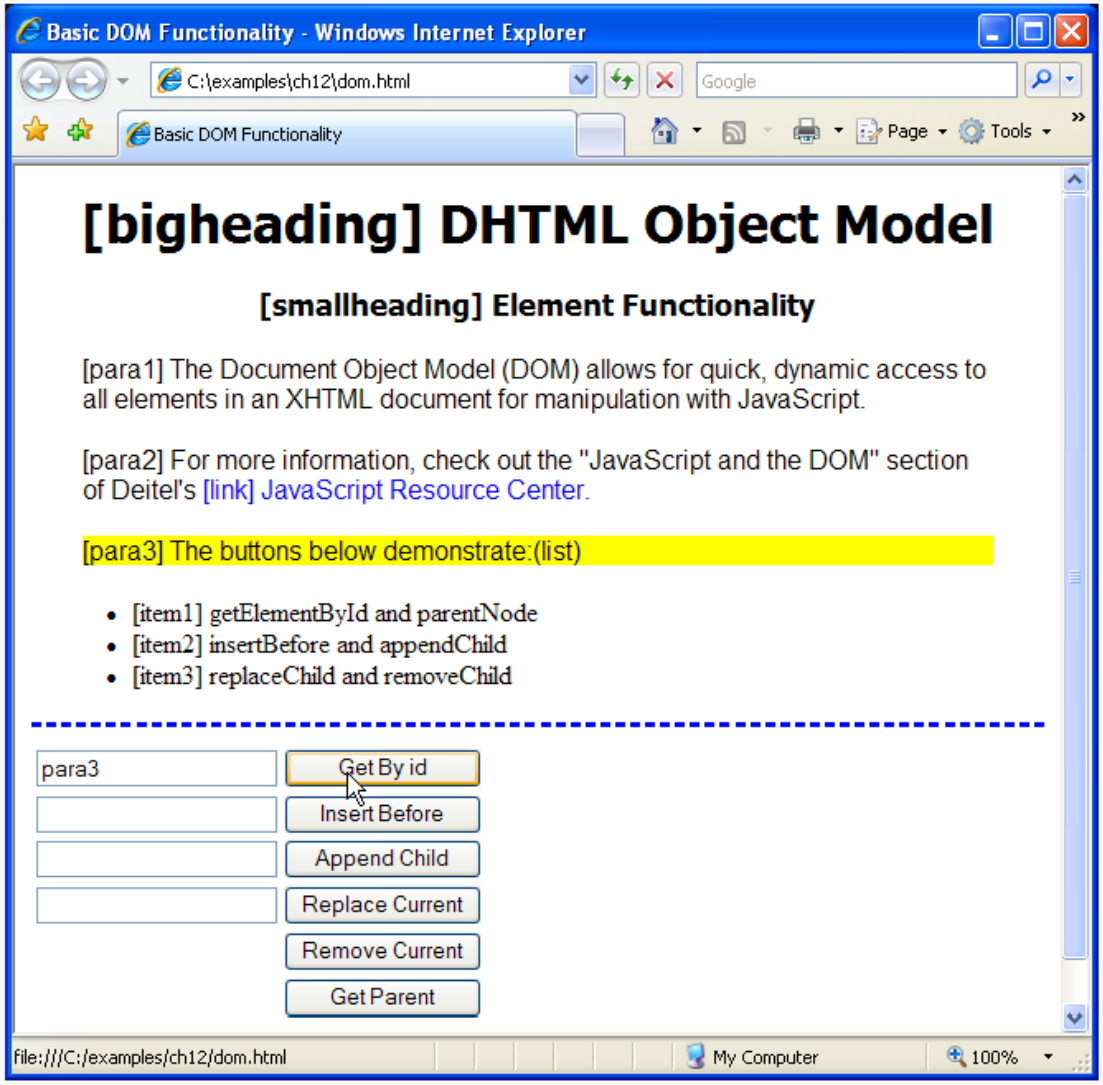

Fig. 12.2 | Basic DOM functionality (Part 8 of 14).

c) This is the document after inserting a new paragraph before the selected one.

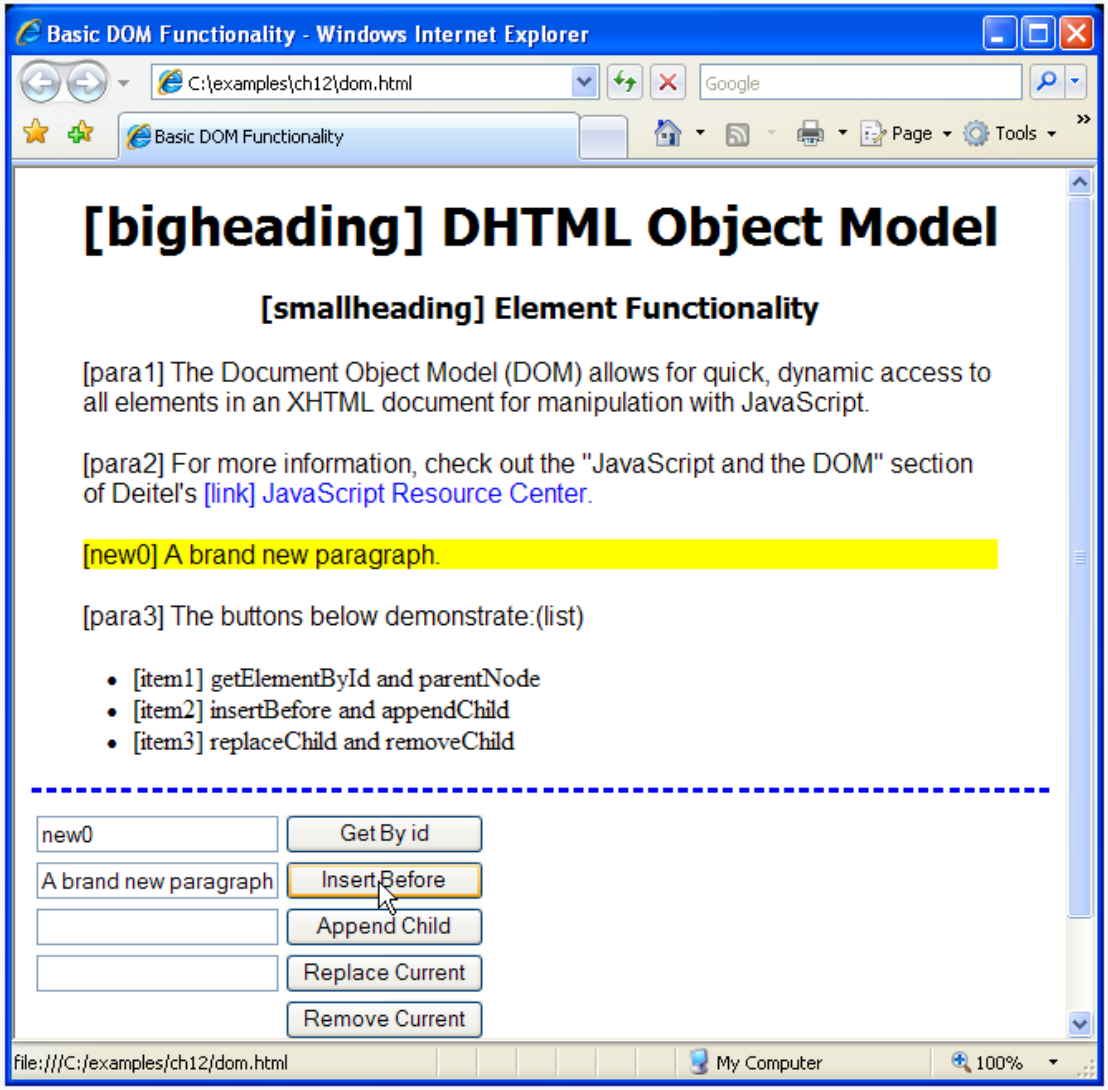

Fig. 12.2 | Basic DOM functionality (Part 9 of 14).

d) Using the Append Child button, a child paragraph is created.

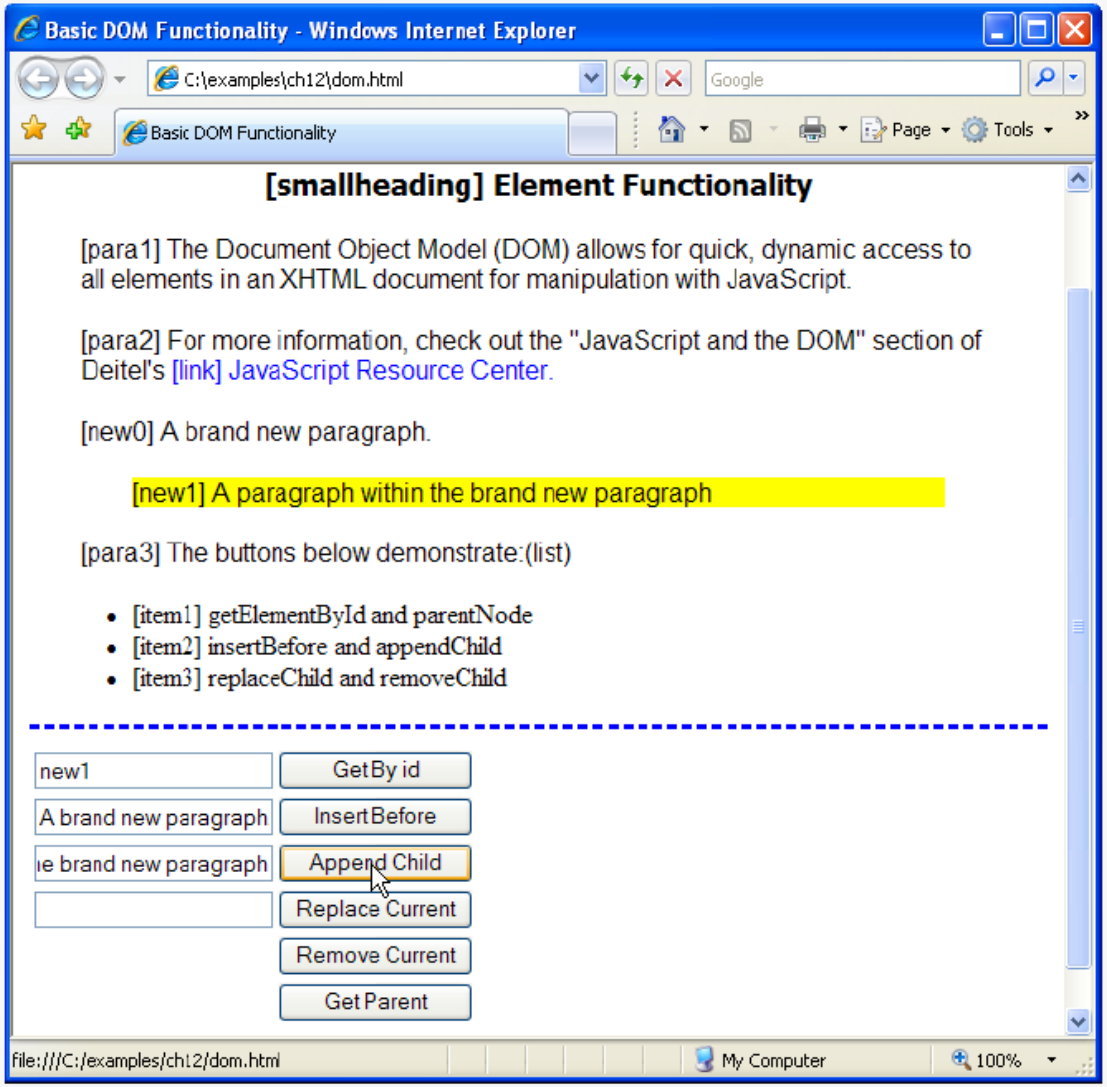

Fig. 12.2 | Basic DOM functionality (Part 10 of 14).

e) The selected paragraph is replaced with a new one.

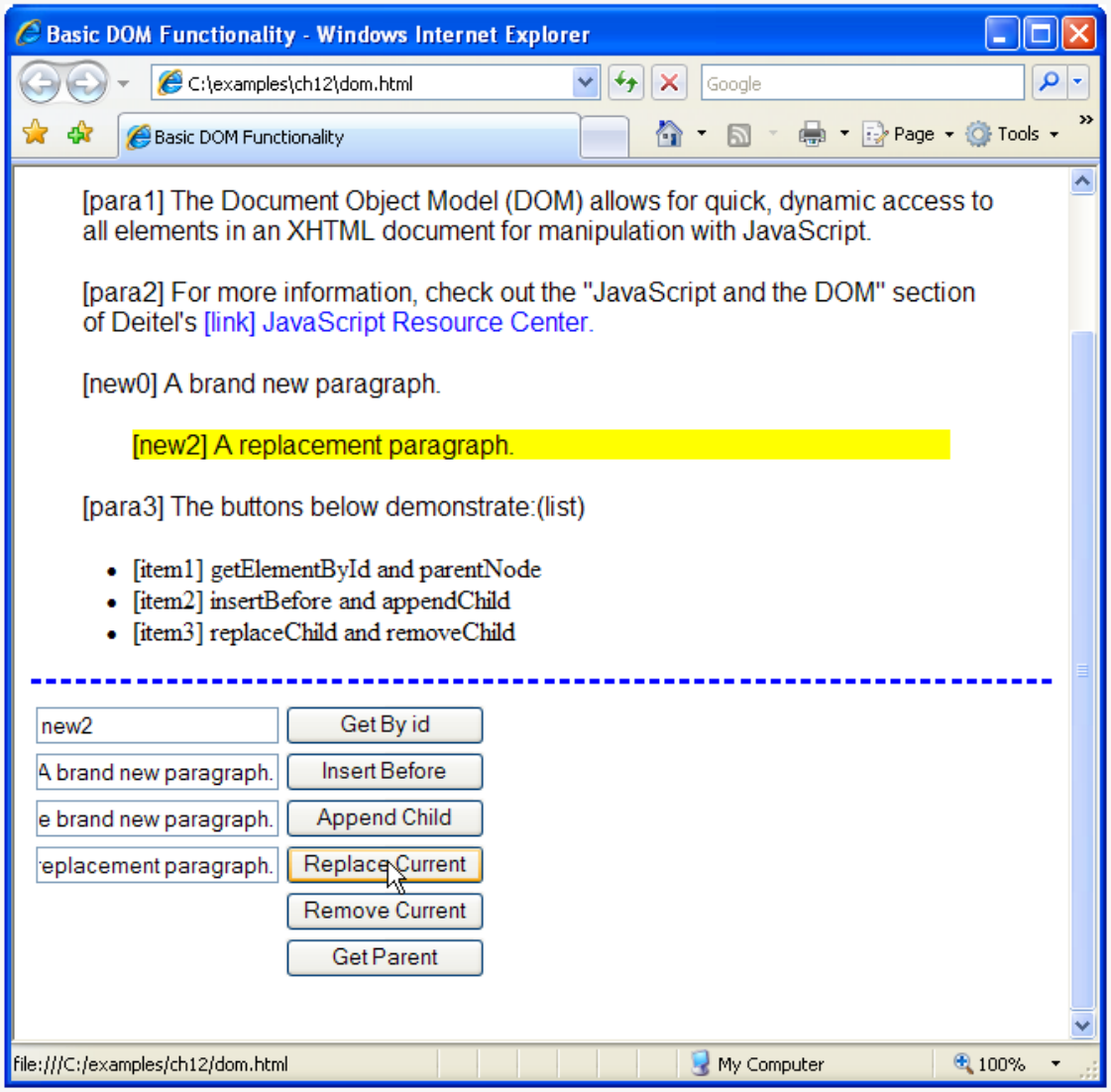

Fig. 12.2 | Basic DOM functionality (Part 11 of 14).

f) The Get Parent button gets the parent of the selected node.

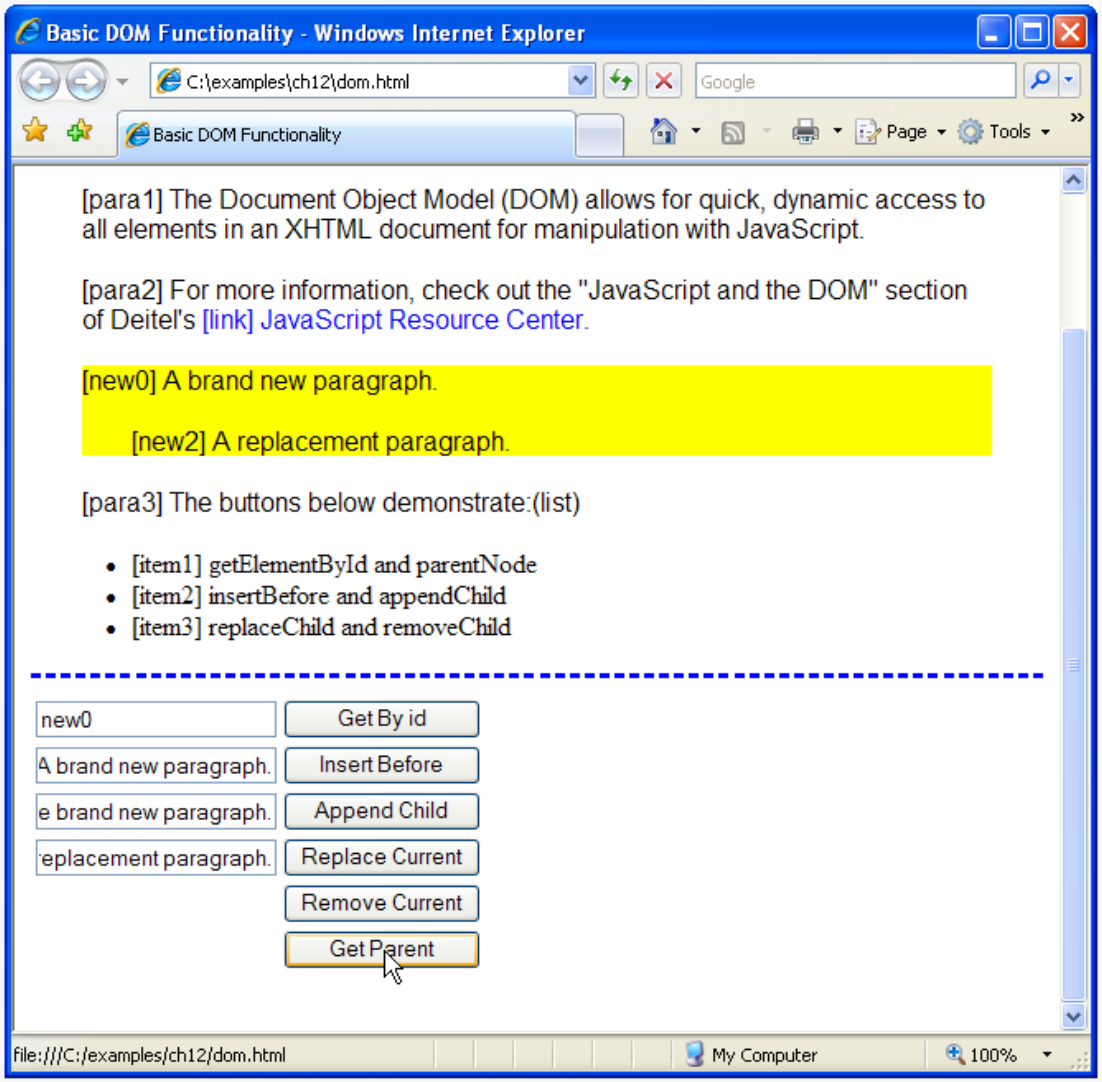

Fig. 12.2 | Basic DOM functionality (Part 12 of 14).

g) Now we select the first list item.

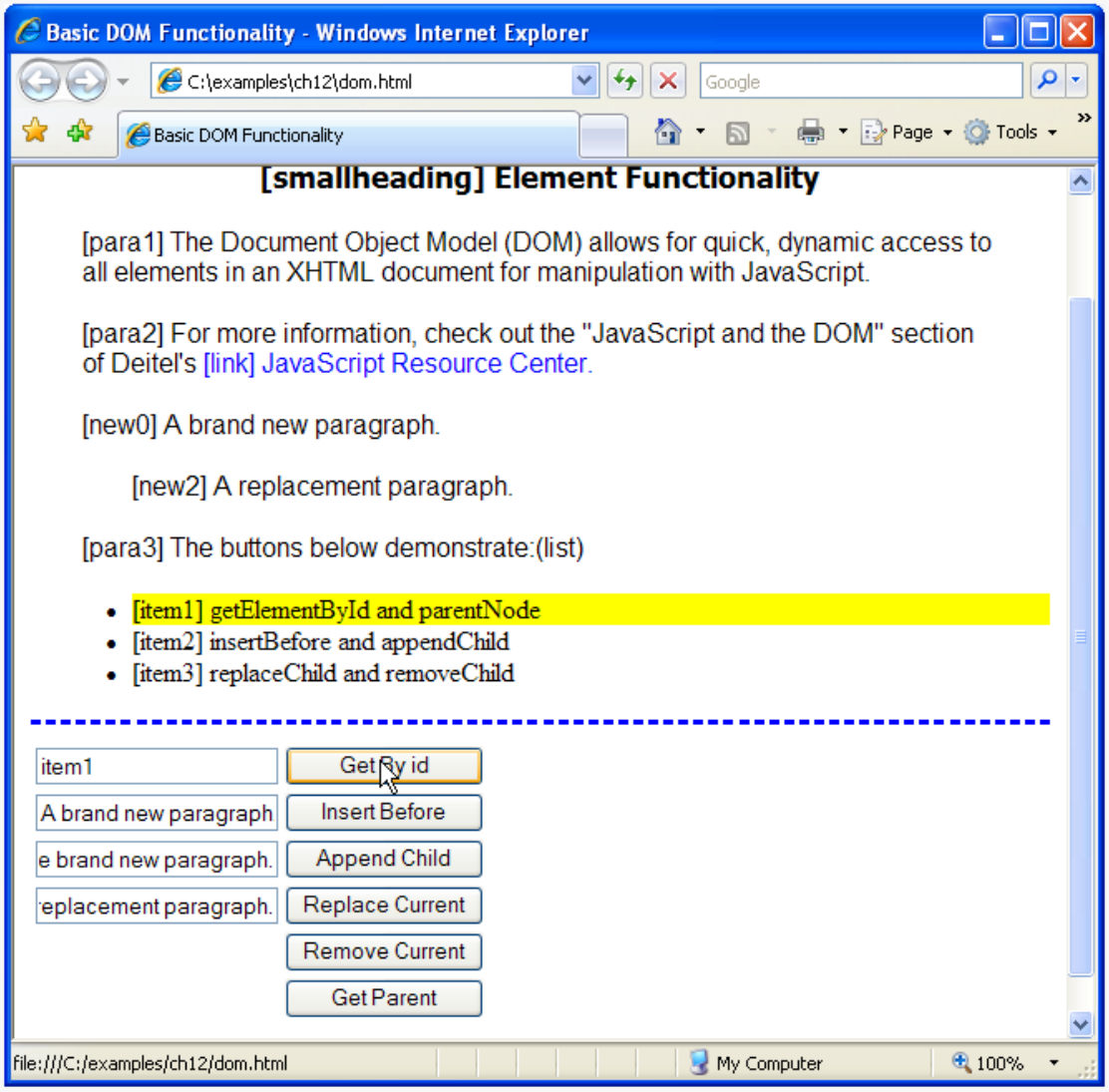

Fig. 12.2 | Basic DOM functionality (Part 13 of 14).

h) The Remove Current button removes the current node and selects its parent.

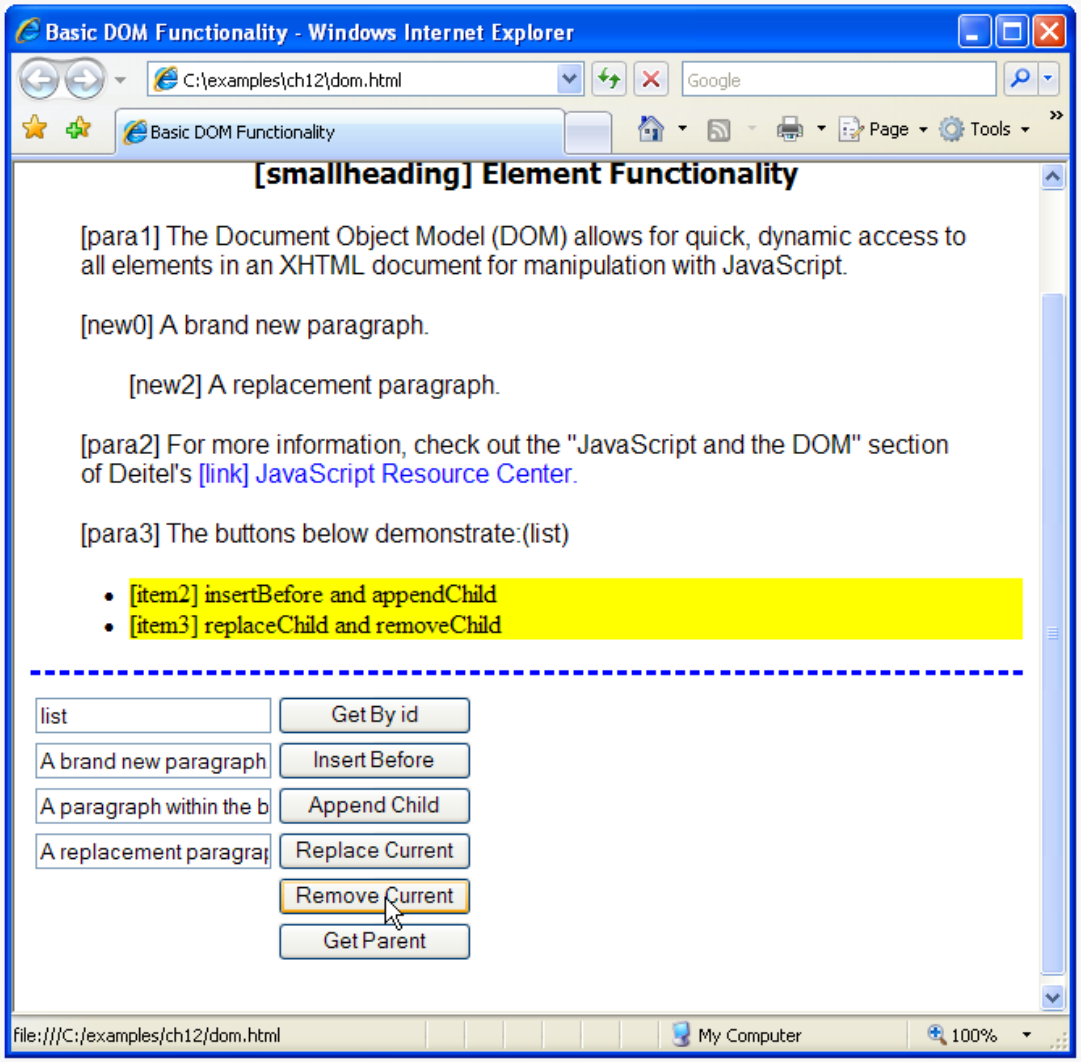

Fig. 12.2 | Basic DOM functionality (Part 14 of 14).

12.4 DOM Collections

- DOM has collections—groups of related objects on a page
- DOM collections are accessed as properties of DOM objects such as the `document` object or a DOM node
- The `document` object has properties containing the `images` collection, `links` collection, `forms` collection and `anchors` collection
  - Contain all the elements of the corresponding type on the page
- To find the number of elements in the collection, use the collection's `length` property

12.4 DOM Collections (Cont.)

- Access items in a collection via square brackets
- `item` method of a DOM collection
  - Access specific elements in a collection, taking an index as an argument
- `namedItem` method
  - takes a name as a parameter and finds the element in the collection, if any, whose `id` attribute or `name` attribute matches it
- `href` property of a DOM link node
  - Refers to the link's `href` attribute
- Collections allow easy access to all elements of a single type in a page
  - Useful for gathering elements into one place and for applying changes across an entire page

Fig. 12.3 | Using the `links` collection (Part 1 of 3).

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 12.3: collections.html -->
6  <!-- Using the links collection. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Using Links Collection</title>
10       <style type = "text/css">
11          body          { font-family: arial, helvetica, sans-serif }
12          h1            { font-family: tahoma, geneva, sans-serif;
13                          text-align: center }
14          p             { margin: 5% }
15          p a           { color: #aa0000 }
16          .links        { font-size: 14px;
17                          text-align: justify;
18                          margin-left: 10%;
19                          margin-right: 10% }
20          .link a       { text-decoration: none }
21          .link a:hover { text-decoration: underline }
22       </style>
23       <script type = "text/javascript">
24          <!--
25          function processlinks()
26          {
27             var linkslist = document.links; // get the document's links
28             var contents = "Links in this page:\n<br />| ";
29
30             // concatenate each link to contents
31             for ( var i = 0; i < linkslist.length; i++ )
```

Stores the document's `links` collection in variable `linkslist`

Number of elements in the collection

Fig. 12.3 | Using the `links` collection (Part 2 of 3).

```
32          {
33             var currentLink = linkslist[ i ];
34             contents += "<span class = 'link'>" +
35                currentLink.innerHTML.link( currentLink.href ) +
36                "</span> | ";
37          } // end for
38
39          document.getElementById( "links" ).innerHTML = contents;
40       } // end function processlinks
41       // -->
42    </script>
43 </head>
44 <body onload = "processlinks()">
45    <h1>Deitel Resource Centers</h1>
46    <p><a href = "http://www.deitel.com/">Deitel's website</a> contains
47       a rapidly growing
48       <a href = "http://www.deitel.com/ResourceCenters.html">list of
49       Resource Centers</a> on a wide range of topics. Many Resource
50       centers related to topics covered in this book,
51       <a href = "http://www.deitel.com/iw3htp4">Internet and World Wide
52       Web How to Program, 4th Edition</a>. We have Resouce Centers on
53       <a href = "http://www.deitel.com/Web2.0">Web 2.0</a>,
54       <a href = "http://www.deitel.com/Firefox">Firefox</a> and
55       <a href = "http://www.deitel.com/IE7">Internet Explorer 7</a>,
56       <a href = "http://www.deitel.com/XHTML">XHTML</a>, and
57       <a href = "http://www.deitel.com/JavaScript">JavaScript</a>.
58       Watch the list of Deitel Resource Centers for related new
59       Resource Centers.</p>
60    <div id = "links" class = "links"></div>
61 </body>
62 </html>
```

Stores the current link in `currentLink`

Uses the `link` method to create an anchor element with proper text and `href` attribute

Puts all links in one location by inserting them into an empty `div` element

The document's links

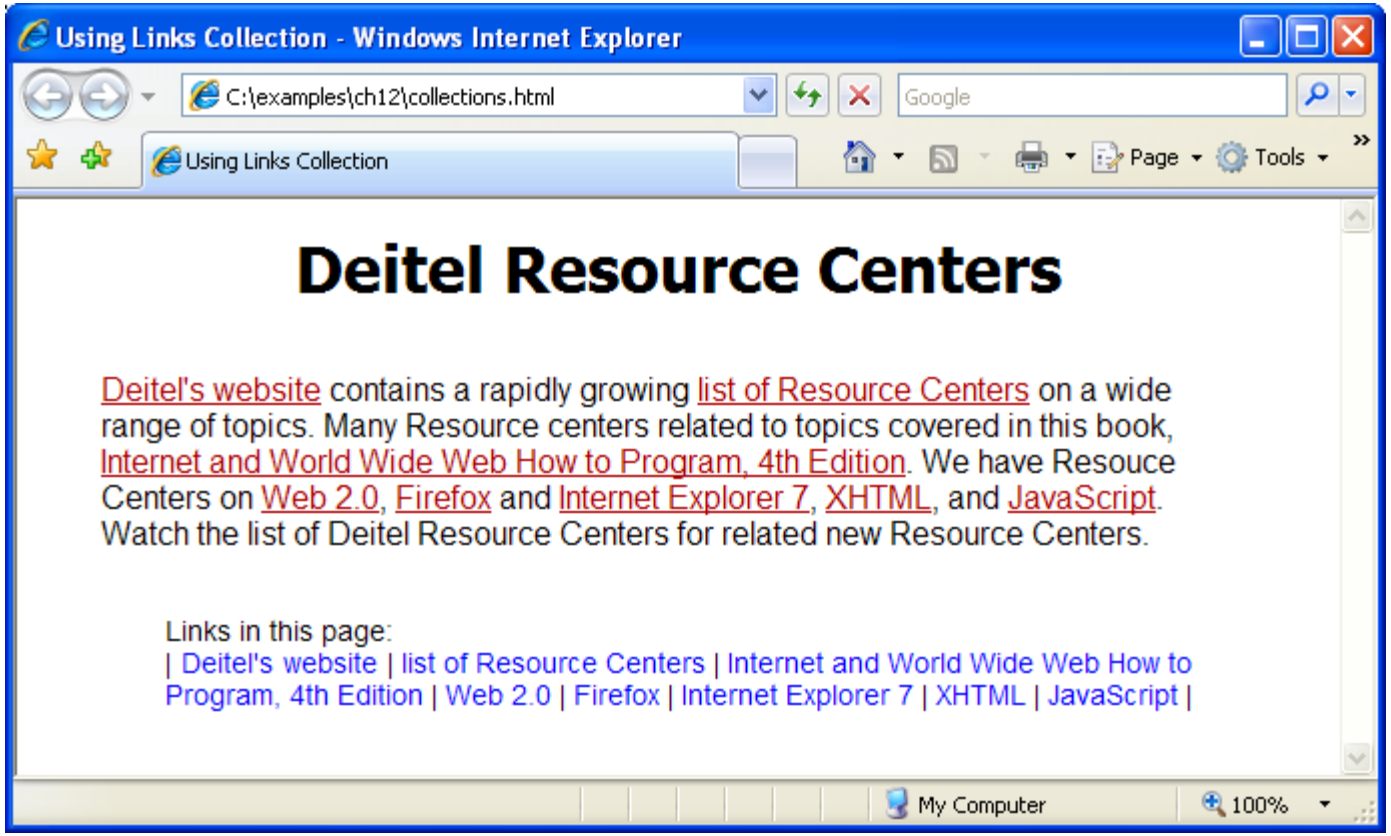

Fig. 12.3 | Using the `links` collection (Part 3 of 3).

12.5 Dynamic Styles

- An element's style can be changed dynamically
  - E.g., in response to user events
  - Can create many effects, including mouse hover effects, interactive menus, and animations
- `body` property of the `document` object
  - Refers to the `body` element in the XHTML page
- `style` property
  - can access a CSS property in the format *node.style.styleproperty*.
- CSS property with a hyphen (`-`), such as `background-color`, is referred to as `backgroundColor` in JavaScript
  - Removing the hyphen and capitalizing the first letter of the following word is the convention for most CSS properties

Fig. 12.4 | Dynamic styles (Part 1 of 2).

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 12.4: dynamicstyle.html -->
6  <!-- Dynamic styles. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Dynamic Styles</title>
10       <script type = "text/javascript">
11          <!--
12          function start()
13          {
14             var inputColor = prompt( "Enter a color name for the " +
15                "background of this page", "" );
16             document.body.style.backgroundColor = inputColor;
17          } // end function start
18          // -->
19       </script>
20    </head>
21    <body id = "body" onload = "start()">
22       <p>Welcome to our website!</p>
23    </body>
24 </html>
```

Prompts the user for a color

Sets the `background-color` CSS property to the user's color

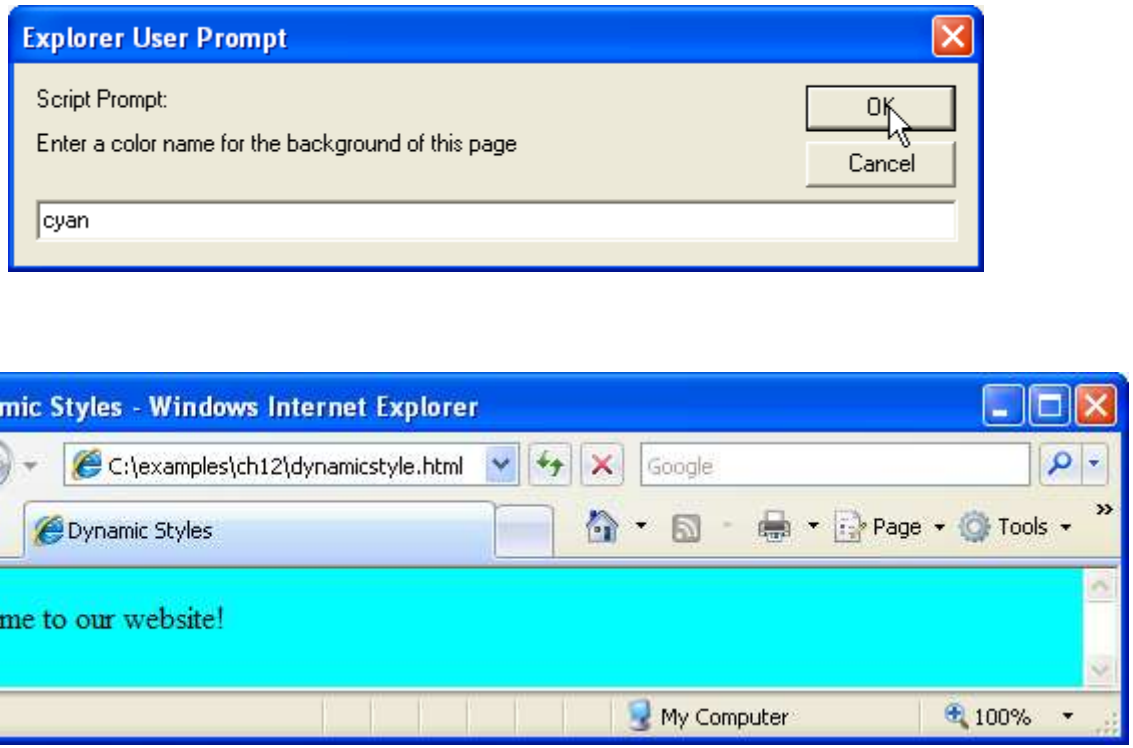

Fig. 12.4 | Dynamic styles (Part 2 of 2).

12.5 Dynamic Styles (Cont.)

- `setInterval` method of the `window` object
  - Repeatedly executes a statement on a certain interval
  - Takes two parameters
    - A statement to execute repeatedly
    - An integer specifying how often to execute it, in milliseconds
  - Returns a unique identifier to keep track of that particular interval.
- `window` object's `clearInterval` method
  - Stops the repetitive calls of object's `setInterval` method
  - Pass to `clearInterval` the interval identifier that `setInterval` returned

```
<?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;
```

Fig. 12.5 | Dynamic styles used for animation (Part 1 of 7).

```
31
32          // stop the animation when the image is large enough
33          if ( count >= 375 )
34          {
35             window.clearInterval( interval );
36             interval = null;
37          } // end if
38
39          var bigImage = document.getElementById( "imgCover" );
40          bigImage.style.width = .7656 * count + "px";
41          bigImage.style.height = count + "px";
42       } // end function run
43
44       // inserts the proper image into the main image area and
45       // begins the animation
46       function display( imgfile )
47       {
48          if ( interval )
49             return;
50
51          var bigImage = document.getElementById( "imgCover" );
52          var newNode = document.createElement( "img" );
53          newNode.id = "imgCover";
54          newNode.src = "fullsize/" + imgfile;
55          newNode.alt = "Large image";
56          newNode.className = "imgCover";
57          newNode.style.width = "0px";
58          newNode.style.height = "0px";
59          bigImage.parentNode.replaceChild( newNode, bigImage );
60          count = 0; // start the image at size 0
```

Stops the animation when the image has reached its full size

Keeps aspect ratio consistent

Sets properties for the new `img` node

Swaps `newNode` for the old cover node

Fig. 12.5 | Dynamic styles used for animation (Part 2 of 7).

```
61            interval = window.setInterval( "run()", 10 ); // animate
62         } // end function display
63         // -->
64      </script>
65   </head>
66   <body>
67      <div id = "mainimg" class = "mainimg">
68         <img id = "imgCover" src = "fullsize/iw3htp4.jpg"
69            alt = "Full cover image" class = "imgCover" />
70      </div>
71      <div id = "thumbs" class = "thumbs" >
72         <img src = "thumbs/iw3htp4.jpg" alt = "iw3htp4"
73            onclick = "display( 'iw3htp4.jpg' )" />
74         <img src = "thumbs/chtp5.jpg" alt = "chtp5"
75            onclick = "display( 'chtp5.jpg' )" />
76         <img src = "thumbs/cpphtp6.jpg" alt = "cpphtp6"
77            onclick = "display( 'cpphtp6.jpg' )" />
78         <img src = "thumbs/jhtp7.jpg" alt = "jhtp7"
79            onclick = "display( 'jhtp7.jpg' )" />
80         <img src = "thumbs/vbhtp3.jpg" alt = "vbhtp3"
81            onclick = "display( 'vbhtp3.jpg' )" />
82         <img src = "thumbs/vcsharphtp2.jpg" alt = "vcsharphtp2"
83            onclick = "display( 'vcsharphtp2.jpg' )" />
84      </div>
85   </body>
86 </html>
```

Executes function `run` every 10 milliseconds

Fig. 12.5 | Dynamic styles used for animation (Part 3 of 7).

a) The cover viewer page loads with the cover of this book.

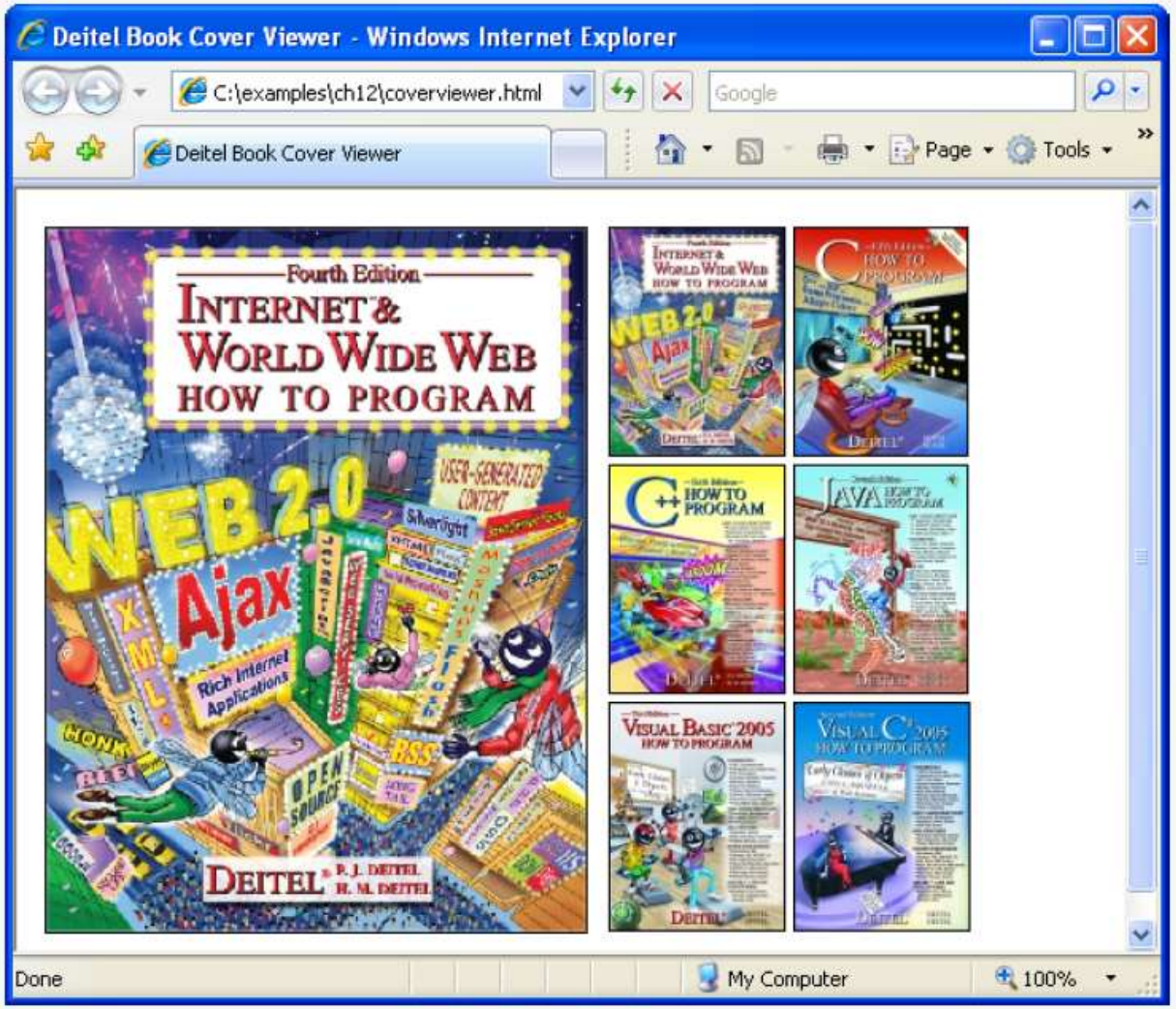

Fig. 12.5 | Dynamic styles used for animation (Part 4 of 7).

b) When the user clicks the thumbnail of C How to Program, the full-size image begins growing from the top-left corner of the window.

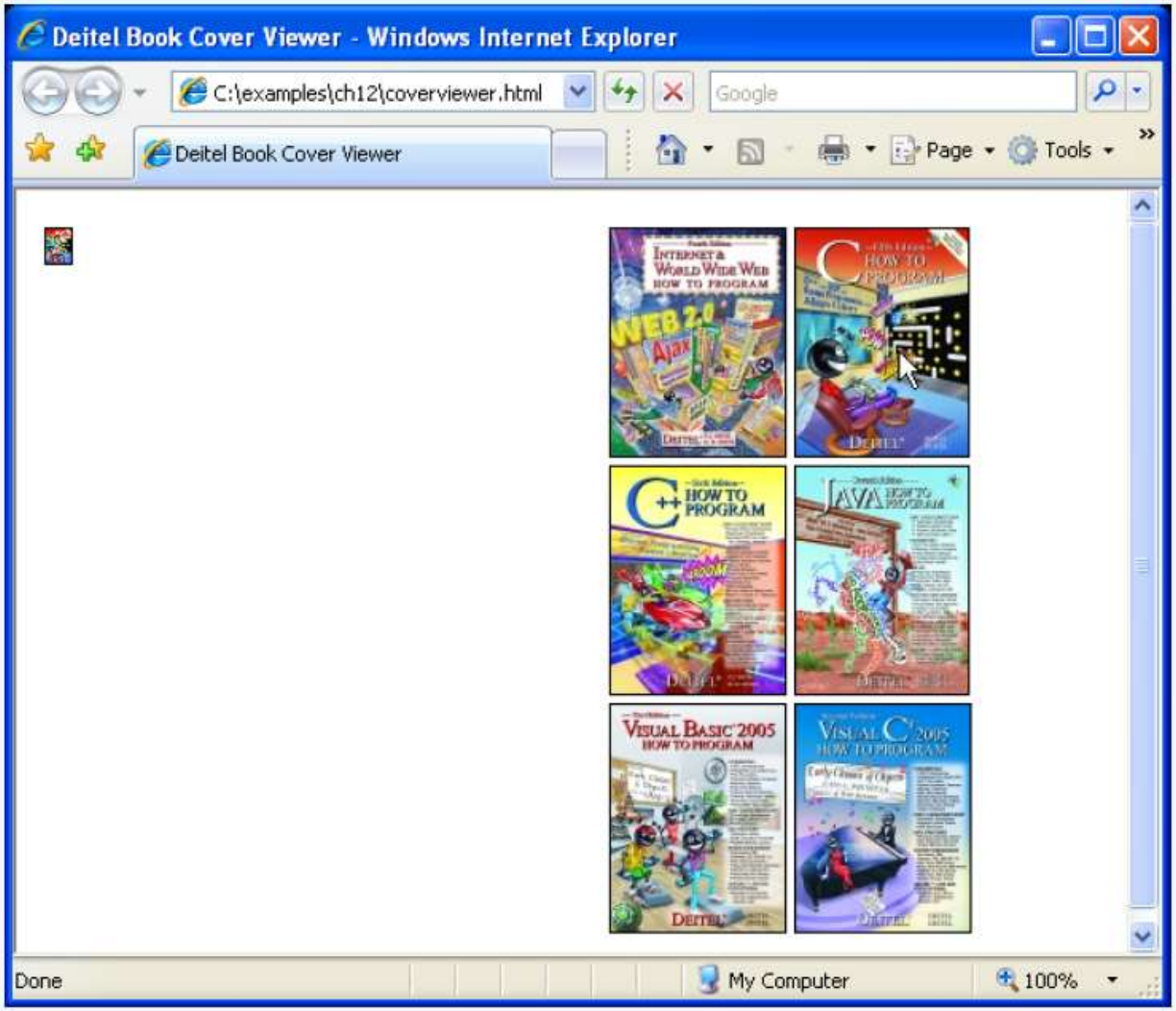

Fig. 12.5 | Dynamic styles used for animation (Part 5 of 7).

c) The cover continues to grow.

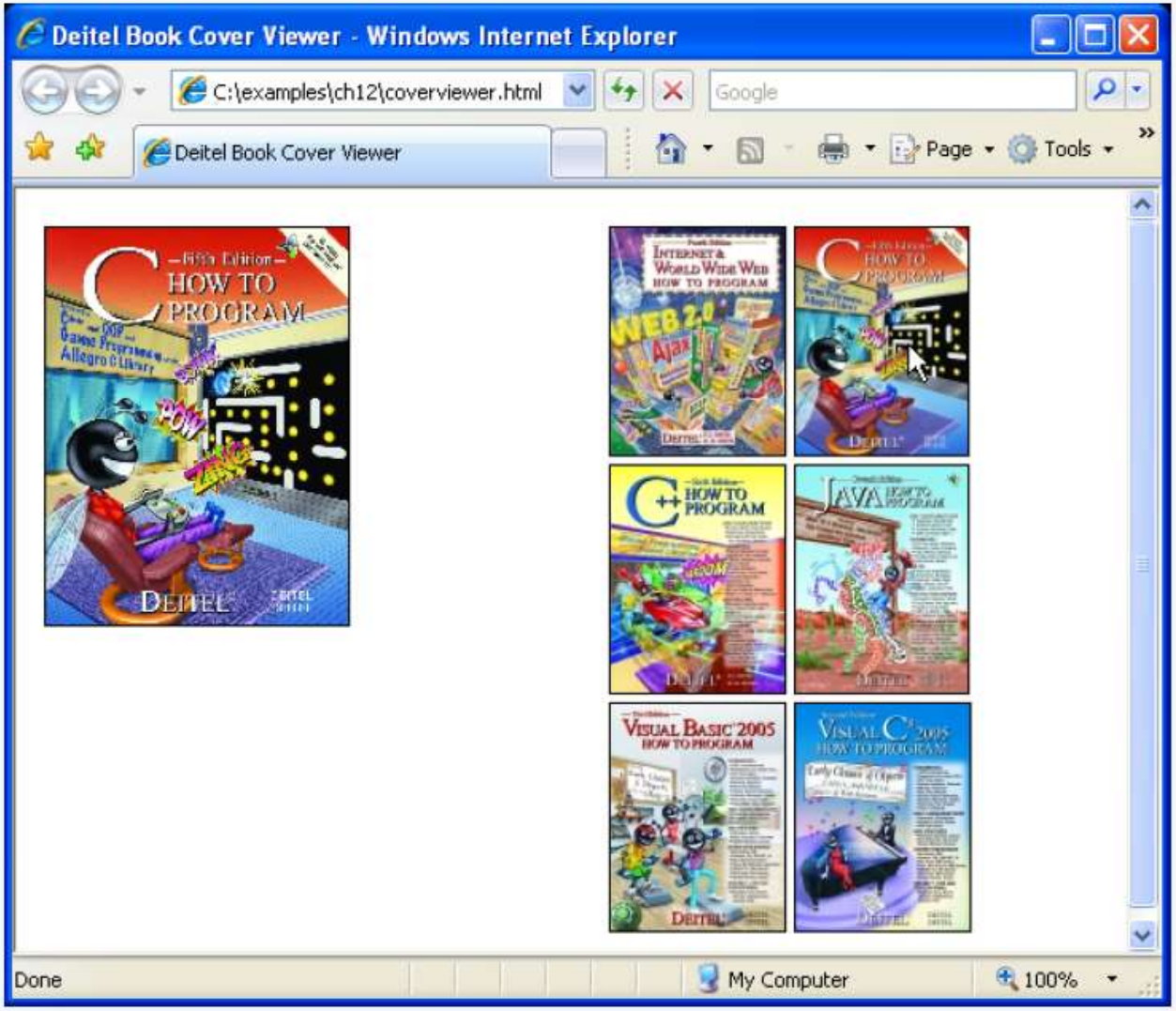

Fig. 12.5 | Dynamic styles used for animation (Part 6 of 7).

d) The animation finishes when the cover reaches its full size.

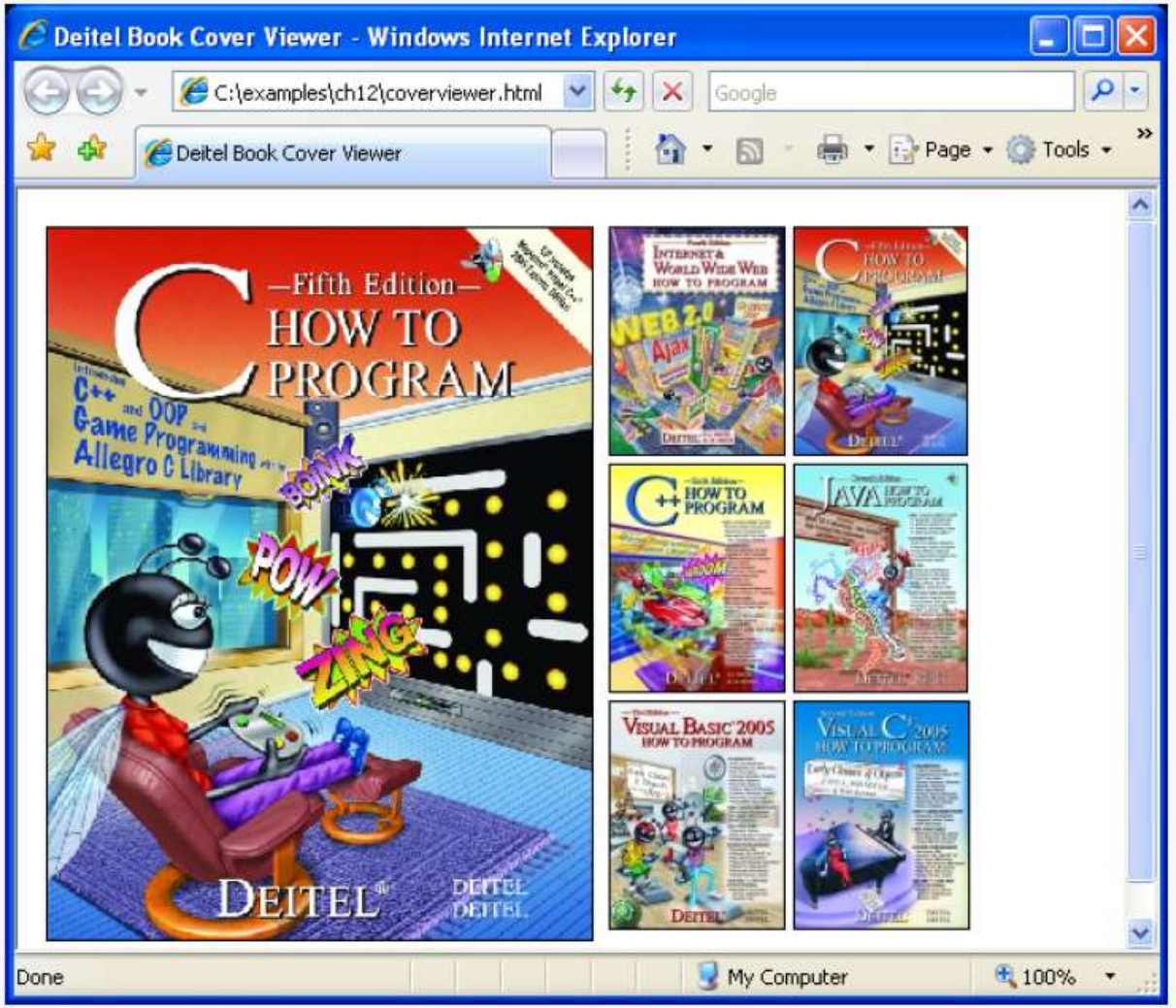

Fig. 12.5 | Dynamic styles used for animation (Part 7 of 7).

12.6 Summary of the DOM Objects and Collections

- **The objects and collections in the W3C DOM give you flexibility in manipulating the elements of a web page.**
- **The W3C DOM allows you to access every element in an XHTML document. Each element in a document is represented by a separate object.**
- **For a reference on the W3C Document Object Model, see the DOM Level 3 recommendation from the W3C at `http://www.w3.org/TR/DOM-Level-3-Core/`. The DOM Level 2 HTML Specification, available at `http://www.w3.org/TR/DOM-Level-2-HTML/`, describes additional DOM functionality specific to HTML, such as objects for various types of XHTML elements.**
- **Not all web browsers implement all features included in the DOM specification.**

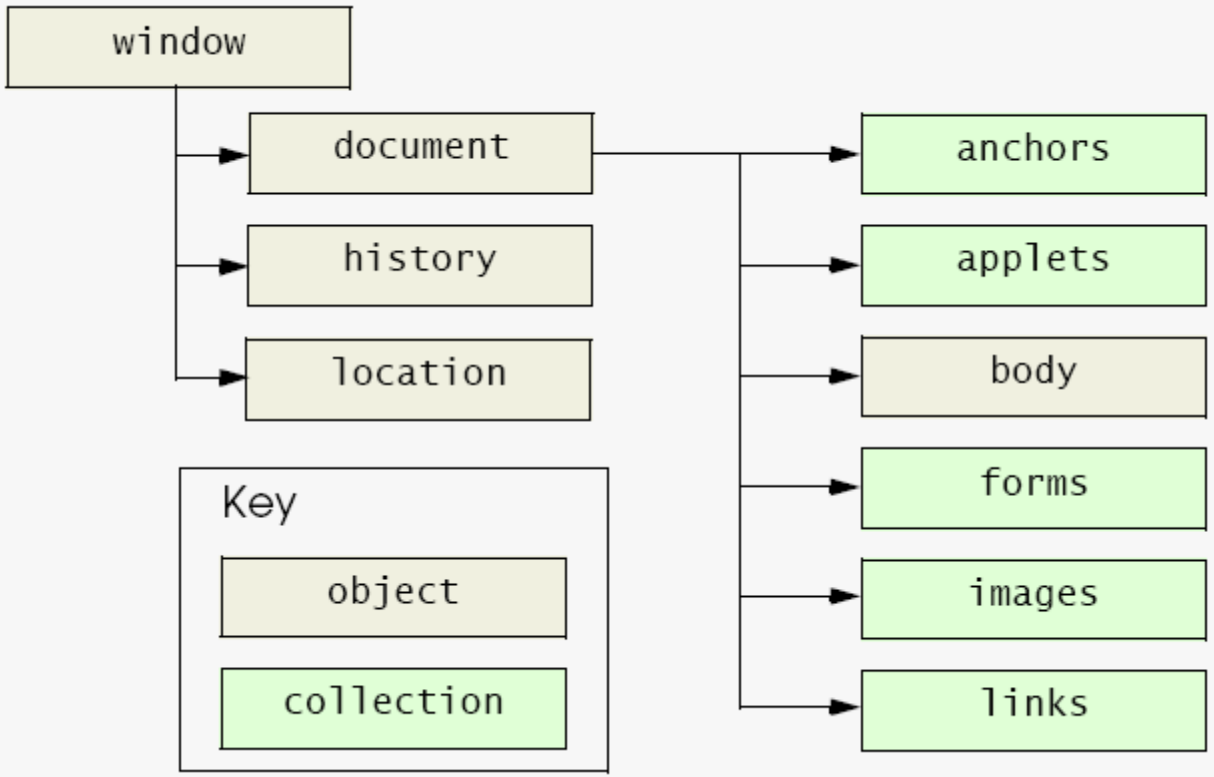

Fig. 12.6 | W3C Document Object Model.

Fig. 12.7 | Objects and collections in the W3C Document Object Model (Part 1 of 2).

Object or collection	Description
Objects	
`window`	Represents the browser window and provides access to the `document` object contained in the `window`. Also contains `history` and `location` objects.
`document`	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.
`body`	Provides access to the `body` element of an XHTML document.
`history`	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.
`location`	Contains the URL of the rendered document. When this object is set to a new URL, the browser immediately navigates to the new location.

Fig. 12.7 | Objects and collections in the W3C Document Object Model (Part 2 of 2).

Object or collection	Description
Collections	
`anchors`	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.
`forms`	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.
`images`	Contains all the `img` elements in the XHTML document. The elements appear in the collection in the order in which they were defined in the XHTML document.
`links`	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.