5.6 Range checks for element inputs can be represented as new properties of the object that represents the element. Modify the document in Exercise 5.5 to add a `max` property value of 99 and a `min` property value of 0. Your event handler must use the properties for the range checks on values input through the text boxes.

5.7 Develop, test, and validate an XHTML document that collects the following information from the user: last name, first name, middle initial, age (restricted to be greater than 17), and weight (restricted to the range from 80 to 300). You must have event handlers for the form elements that collect this information. These handlers must check the input data for correctness. Messages in `alert` windows must be produced when errors are detected.

5.8 Revise the document of Exercise 5.1 to use the DOM 2 event model.

5.9 Revise the document of Exercise 5.3 to use the DOM 2 event model.

CHAPTER 6
Dynamic Documents with JavaScript

Informally, a dynamic XHTML document is an XHTML document that, in some way, can be changed while it is being displayed by a browser. The most common client-side approach to providing dynamic documents is to use JavaScript to manipulate the objects of the Document Object Model (DOM) of the displayed document. Changes to documents can occur when they are explicitly requested by user interactions, at regular timed intervals, or when browser events occur.

XHTML elements can be initially positioned at any given location on the browser display. If they're positioned in a specific way, elements can be dynamically moved to new positions on the display. Elements can be made to disappear and reappear. The colors of the background and the foreground (the elements) of a document can be changed. The font, font size, and font style of displayed text can be changed. Even the content of an element can be changed. Overlapping elements in a document can be positioned in a specific top-to-bottom stacking order, and their stacking order can be dynamically changed. The position of the mouse cursor on the browser display can be determined when a mouse button is clicked. Elements can be made to move slowly around the display screen. Finally, elements can be defined to allow the user to drag and drop them anywhere in the display window. This chapter discusses the JavaScript code that can create all of these effects.

6.1 Introduction

Dynamic XHTML is not a new markup language. It is a collection of technologies that allows dynamic changes to documents defined with XHTML. Specifically, a *dynamic XHTML document* is an XHTML document whose tag attributes, tag contents, or element style properties can be changed by user interaction or the occurrence of a browser event after the document has been, and is still being, displayed. Such changes can be made with an embedded script that accesses the elements of the document as objects in the associated DOM structure.

Support for dynamic XHTML is not uniform across the various browsers. As in Chapter 5, "JavaScript and XHTML Documents," the discussion here is restricted to W3C-standard approaches rather than including features defined by a particular browser vendor. All of the examples in this chapter, except the document in Section 6.11, use the DOM 0 event model and work on both Internet Explorer 8 (IE8) and Firefox 3 (FX3) browsers. The example in Section 6.11 uses the DOM 2 event model because it cannot be designed in a standard way with the DOM 0 event model. Because IE8 does not support the DOM 2 event model, that example does not work with IE8.

This chapter discusses user interactions through XHTML documents using client-side JavaScript. Chapters 8–10 discuss user interactions through XHTML documents using server-side technologies.

6.2 Positioning Elements

Before the browsers that implemented HTML 4.0 appeared, Web site authors had little control over how HTML elements were arranged in documents. In many cases

Before the browsers that implemented HTML 4.0 appeared, Web site authors had little control over how HTML elements were arranged in documents. In many cases, the elements found in the HTML file were simply placed in the document the way text is placed in a document with a word processor: Fill a row, start a new row, fill it, and so forth. HTML tables provide a framework of columns for arranging elements, but they lack flexibility and also take a considerable time to display.[1] This lack of powerful and efficient element placement control ended when Cascading Style Sheets–Positioning (CSS-P) was released by the W3C in 1997.

CSS-P is completely supported by IE8 and FX3. It provides the means not only to position any element anywhere in the display of a document, but also to move an element to a new position in the display dynamically, using JavaScript to change the positioning style properties of the element. These style properties, which are appropriately named `left` and `top`, dictate the distance from the left and top of some reference point to where the element is to appear. Another style property, `position`, interacts with `left` and `top` to provide a higher level of control of placement and movement of elements. The `position` property has three possible values: `absolute`, `relative`, and `static`.

6.2.1 Absolute Positioning

The `absolute` value is specified for `position` when the element is to be placed at a specific place in the document display without regard to the positions of other elements. For example, if a paragraph of text is to appear 100 pixels from the left edge and 200 pixels from the top of the display window, the following element could be used:

```
<p style = "position: absolute; left: 100px; top: 200px">
    -- text --
</p>
```

One use of absolute positioning is to superimpose special text over a paragraph of ordinary text to create an effect similar to a watermark on paper. A larger italicized font, in a light-gray color and with space between the letters, could be used for the special text, allowing both the ordinary text and the special text to be legible. The XHTML document that follows provides an example that implements this effect. In this example, a paragraph of normal text that describes apples is displayed. Superimposed on this paragraph is the somewhat subliminal message “APPLES ARE GOOD FOR YOU”. Here is the document:

```
<?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">

<!-- absPos.html
     Illustrates absolute positioning of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Absolute positioning </title>
    <style type = "text/css">

/* A style for a paragraph of text */
    .regtext {font-family: Times; font-size: 14pt; width: 600px}

/* A style for the text to be absolutely positioned */
    .abstext {position: absolute; top: 25px; left: 50px;
              font-family: Times; font-size: 24pt;
              font-style: italic; letter-spacing: 1em;
              color: rgb(102,102,102); width: 500px}
    </style>
  </head>
  <body>
    <p class = "regtext">
      Apple is the common name for any tree of the genus Malus,
      of the family Rosaceae. Apple trees grow in any of the
      temperate areas of the world. Some apple blossoms are white,
      but most have stripes or tints of rose. Some apple blossoms
      are bright red. Apples have a firm and fleshy structure that
      grows from the blossom. The colors of apples range from
      green to very dark red. The wood of apple trees is fine
      grained and hard. It is, therefore, good for furniture
      construction. Apple trees have been grown for many
      centuries. They are propogated by grafting because they
      do not reproduce themselves.
    </p>
    <p class = "abstext">
      APPLES ARE GOOD FOR YOU
    </p>
  </body>
</html>
```

Figure 6.1 shows a display of `absPos.html`.

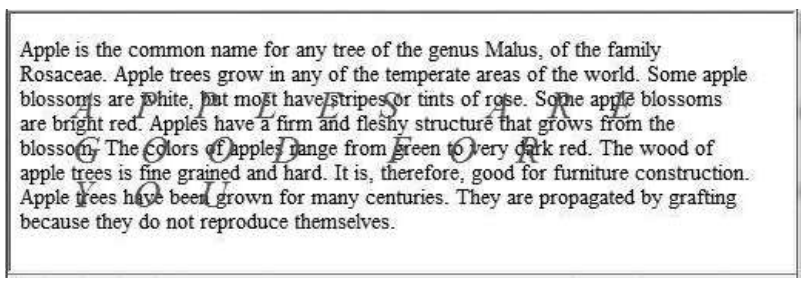

Figure 6.1 Display of `absPos.html`

Notice that a `width` property value is included in the style for both the regular and the special text. This property is used here to ensure that the special text is uniformly embedded in the regular text. Without it, the text would extend to the right end of the browser display window—and, of course, the width of the window could vary widely from client to client and even from minute to minute on the same client because the user can resize the browser window at any time.

When an element is absolutely positioned inside another positioned element (one that has the `position` property specified), the `top` and `left` property values are measured from the upper-left corner of the enclosing element (rather than the upper-left corner of the browser window).

To illustrate the placement of nested elements, the document `absPos.html` is modified to place the regular text 100 pixels from the top and 100 pixels from the left. The special text is nested inside the regular text by using `<div>` and `<span>` tags. The modified document, which is named `absPos2.html`, is as follows:

```
<?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">

<!-- absPos2.html
     Illustrates nested absolute positioning of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Nested absolute positioning </title>
    <style type = "text/css">

/* A style for a paragraph of text */
     .regtext {font-family: Times; font-size: 14pt; width: 500px;
               position: absolute; top: 100px; left: 100px;}

/* A style for the text to be absolutely positioned */
     .abstext {position: absolute; top: 25px; left: 50px;
               font-family: Times; font-size: 24pt;
               font-style: italic; letter-spacing: 1em;
               color: rgb(102,102,102); width: 400px;}
    </style>
  </head>
  <body>
    <div class = "regtext">
      Apple is the common name for any tree of the genus Malus,
      of the family Rosaceae. Apple trees grow in any of the
      temperate areas of the world. Some apple blossoms are white,
      but most have stripes or tints of rose. Some apple blossoms
      are bright red. Apples have a firm and fleshy structure that
      grows from the blossom. The colors of apples range from
      green to very dark red. The wood of apple trees is fine
      grained and hard. It is, therefore, good for furniture
      construction. Apple trees have been grown for many
      centuries. They are propagated by grafting because they
      do not reproduce themselves.
      <span class = "abstext">
        APPLES ARE GOOD FOR YOU
      </span>
    </div>
  </body>
</html>
```

Figure 6.2 shows a display of `absPos2.html`.

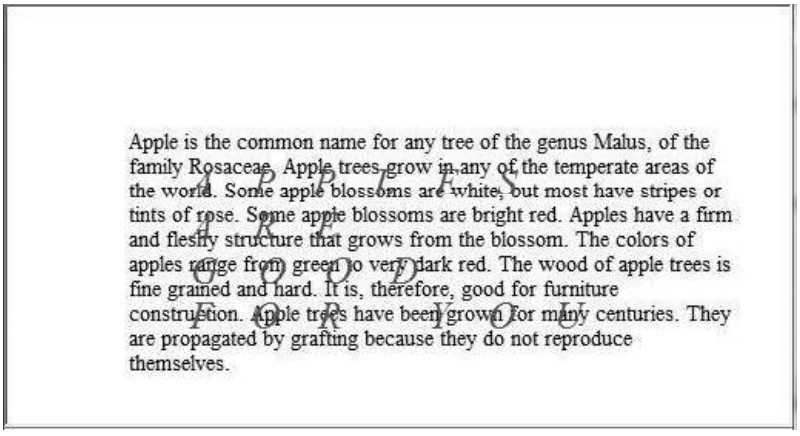

Figure 6.2 Display of `absPos2.html`

6.2.2 Relative Positioning

An element that has the `position` property set to `relative`, but does not specify `top` and `left` property values, is placed in the document as if the `position` attribute were not set at all. However, such an element can be moved later. If the `top` and `left` properties are given values, they displace the element

by the specified amount from the position where it would have been placed (if top and left had not been set). For example, suppose that two buttons are placed in a document and the position attribute has its default value, which is static. Then the buttons would appear next to each other in a row, assuming that the current row has sufficient horizontal space for them. If position has been set to relative and the second button has its left property set to 50px, the effect would be to move the second button 50 pixels farther to the right than it otherwise would have appeared.

In both the case of an absolutely positioned element inside another element and the case of a relatively positioned element, negative values of top and left displace the element upward and to the left, respectively.[2]

Relative positioning can be used for a variety of special effects in placing elements. For example, it can be used to create superscripts and subscripts by placing the values to be raised or lowered in tags and displacing them from their regular positions. In the next example, a line of text is set in a normal font style in 24-point size. Embedded in the line is one word that is set in italic, 48-point, red font. Normally, the bottom of the special word would align with the bottom of the rest of the line. In this case, the special word is to be vertically centered in the line, so its position property is set to relative and its top property is set to 10 pixels, which lowers it by that amount relative to the surrounding text. The XHTML document to specify this, which is named relPos.html, is as follows:

```
<?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">

<!-- relPos.html
     Illustrates relative positioning of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Relative positioning </title>
  </head>
  <body style = "font-family: Times; font-size: 24pt;">
    <p>
      Apples are <span style =
                "position: relative; top: 10px;
                 font-family: Times; font-size: 48pt;
                 font-style: italic; color: red;">
      GOOD </span> for you.
    </p>
  </body>
</html>
```

Figure 6.3 shows a display of relPos.html.

Figure 6.3 Display of relPos.html

6.2.3 Static Positioning

The default value for the position property is static. A statically positioned element is placed in the document as if it had the position value of relative but no values for top or left were given. The difference is that a statically positioned element cannot have its top or left properties initially set or changed later. Therefore, a statically placed element cannot be displaced from its normal position and cannot be moved from that position later.

6.3 Moving Elements

As stated previously, an XHTML element whose position property is set to either absolute or relative can be moved. Moving an element is simple: Changing the top or left property values causes the element to move on the display. If its position is set to absolute, the element moves to the new values of top and left; if its position is set to relative, it moves from its original position by distances given by the new values of top and left.

In the next example, an image is absolutely positioned in the display. The document includes two text boxes, labeled x coordinate and y coordinate, respectively. The user can enter new values for the left and top properties of the image in these boxes. When the button labeled *Move It* is pressed, the values of the left and top properties of the image are changed to the given values, and the element is moved to its new position.

A JavaScript function, stored in a separate file, is used to change the values of left and top in this example. Although it is not necessary here, the id of the element to be moved is sent to the function that does the moving, just to illustrate that the function could be used on any number of different elements. The values of the two text boxes are also sent to the function as parameters. The actual parameter values are the DOM addresses of the text boxes, with the value attribute attached, which provides the complete DOM addresses of the text box values. Notice that style is attached to the DOM address of the image to be moved because top and left are style properties. Because the input top and left values from the text boxes are just string representations of numbers, but the top and left properties must end with some unit abbreviation, the event handler catenates "px" to each value before assigning it to the top and left properties. This document, called mover.html, and the associated JavaScript file, mover.js, are as follows:

```
<?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">

<!-- mover.html
     Uses mover.js to move an image within a document
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Moving elements </title>
    <script type = "text/javascript"  src = "mover.js" >
    </script>
  </head>
  <body>
    <form action = "">
      <p>
        <label>
          x coordinate:
          <input type = "text"  id = "leftCoord" size = "3" />
        </label>
        <br />
        <label>
          y coordinate:
          <input type = "text"  id = "topCoord" size = "3" />
        </label>
        <br />
        <input type = "button"  value = "Move it"
               onclick =
                 "moveIt('saturn',
                 document.getElementById('topCoord').value,
                 document.getElementById('leftCoord').value)" />
      </p>
    </form>
    <div id = "saturn"  style = "position: absolute;
         top: 115px; left: 0;">
      <img src = "../images/ngc604.jpg"
           alt = "(Pictures of Saturn)" />
    </div>
  </body>
</html>
```

```
// mover.js
//   Illustrates moving an element within a document

// The event handler function to move an element
function moveIt(movee, newTop, newLeft) {
  dom = document.getElementById(movee).style;

// Change the top and left properties to perform the move
//  Note the addition of units to the input values
  dom.top = newTop + "px";
  dom.left = newLeft + "px";
}
```

Figures 6.4 and 6.5 respectively show the initial and new positions of an image in `mover.html`.

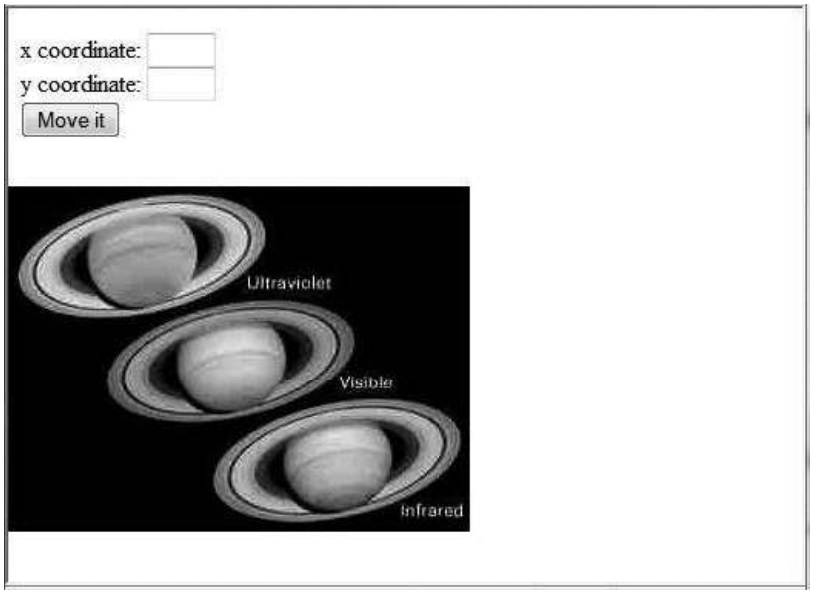

Figure 6.4 Display of `mover.html` (before pressing the *Move It* button)

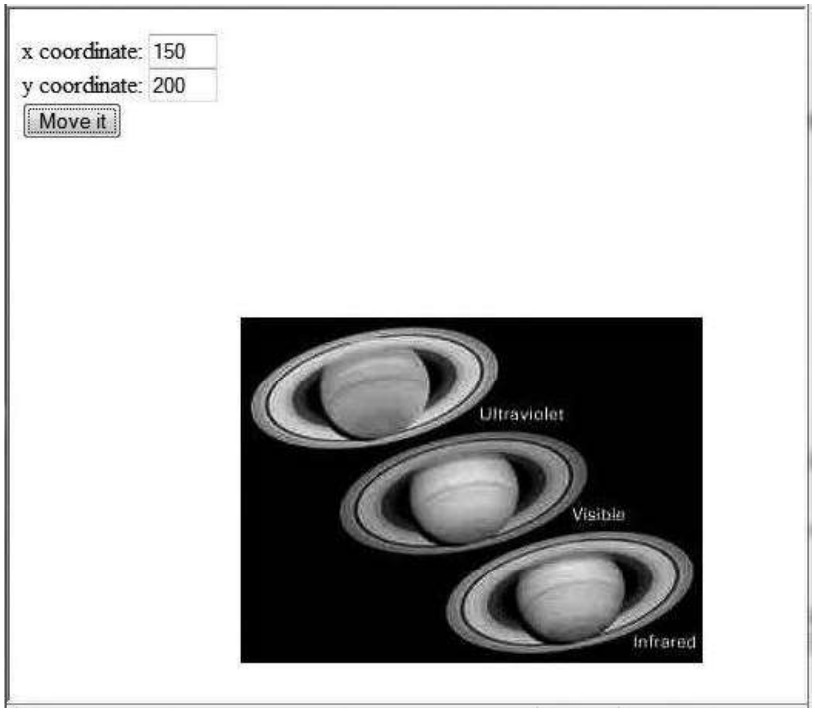

Figure 6.5 Display of `mover.html` (after pressing the *Move It* button)

6.4 Element Visibility

Document elements can be specified to be visible or hidden with the value of their `visibility` property. The two possible values for `visibility` are, quite naturally, `visible` and `hidden`. The appearance or disappearance of an element can be controlled by the user through a widget.

The following example displays an image and allows the user to toggle a button, causing the image to appear and not appear in the document display (once again, the event handler is in a separate file):

```
<?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">

<!-- showHide.html
      Uses showHide.js
      Illustrates visibility control of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Visibility control </title>
    <script type = "text/javascript"  src = "showHide.js" >
    </script>
  </head>
  <body>
    <form action = "">
      <div id = "saturn"  style = "position: relative;
           visibility: visible;">
        <img src = "../images/saturn.jpg"
             alt = "(Pictures of Saturn)" />
      </div>
      <p>
        <br />
        <input type = "button"  value = "Toggle Saturn"
               onclick = "flipImag()" />
      </p>
    </form>
  </body>
</html>
```

```
// showHide.js
//   Illustrates visibility control of elements

// The event handler function to toggle the visibility
//   of the images of Saturn
function flipImag() {
  dom = document.getElementById("saturn").style;

// Flip the visibility adjective to whatever it is not now
 if (dom.visibility == "visible")
   dom.visibility = "hidden";
 else
   dom.visibility = "visible";
}
```

6.5 Changing Colors and Fonts

The background and foreground colors of the document display can be dynamically changed, as can the font properties of the text.

6.5.1 Changing Colors

Dynamic changes to colors are relatively simple. In the next example, the user is presented with two text boxes into which color specifications can be typed—one for the document background color and one for the foreground color. The colors can be specified in any of the three ways that color properties can be given in CSS. A JavaScript function that is called whenever one of the text boxes is changed makes the change in the document's appropriate color property: `back-groundColor` or `color`. The first of the two parameters to the function specifies whether the new color is for the background or foreground; the second specifies the new color. The new color is the `value` property of the text box that was changed by the user.

In this example, the calls to the handler functions are in the XHTML text box elements. This approach allows a simple way to reference the element's DOM address. The JavaScript `this` variable in this situation is a reference to the object that represents the element in which it is referenced. A reference to such an object is its DOM address. Therefore, in a text element, the value of `this` is the DOM address of the text element. So, in the example, `this.value` is used as an actual parameter to the handler function. Because the call is in an input element, `this.value` is the DOM address of the value of the input element. This document, called `dynColors.html`, and the associated JavaScript file are as follows:

```
<?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">

<!-- dynColors.html
     Uses dynColors.js
     Illustrates dynamic foreground and background colors
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Dynamic colors </title>
    <script type = "text/javascript"  src = "dynColors.js" >
    </script>
  </head>
  <body>
    <p style = "font-family: Times; font-style: italic;
                font-size: 24pt" >
      This small page illustrates dynamic setting of the
      foreground and background colors for a document
    </p>
    <form action = "">
      <p>
        <label>
          Background color:
          <input type = "text"  name = "background" size = "10"
                 onchange = "setColor('background', this.value)" />
        </label>
        <br />
        <label>
          Foreground color:
          <input type = "text"  name = "foreground" size = "10"
                 onchange = "setColor('foreground', this.value)" />
        </label>
        <br />
      </p>
    </form>
  </body>
</html>
```

```
// dynColors.js
//   Illustrates dynamic foreground and background colors

// The event handler function to dynamically set the
// color of background or foreground
function setColor(where, newColor) {
  if (where == "background")
    document.body.style.backgroundColor = newColor;
  else
    document.body.style.color = newColor;
}
```

6.5.2 Changing Fonts

Web users are accustomed to having links in documents change color when the cursor is placed over them. Use of the `mouseover` event to trigger a JavaScript event handler allows us to change any property of any element in a document, including text, when the mouse cursor is placed over it. Thus, the font style and font size, as well as the color and background color of text, can be changed when the cursor is placed over the text. The text can be changed back to its original form when an event handler is triggered with the `mouseout` event.

For CSS attribute names that are single words without hyphens, the associated JavaScript property names are the same as the attribute names. But when an attribute name includes a hyphen, as in `font-size`, the associated property name must be different (because a property name cannot include a hyphen). The convention is that when an attribute name has a hyphen, the hyphen is deleted and the letter that follows is capitalized in its associated property name. So, the property name associated with the attribute `font-size` is `fontSize`.

In the next example, the only element is a sentence with an embedded special word. The special word is the content of a span element, so its attributes can be changed. The foreground color for the document is the default black. The word is presented in blue. When the mouse cursor is placed over the word, its color changes to red, its font style changes to italic, and its size changes from 16 point to 24 point. When the cursor is moved off the word, it reverts to its original style. Here is this document, called `dynFont.html`:

```
<?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">

<!-- dynFont.html
     Illustrates dynamic font styles and colors
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Dynamic fonts </title>
    <style type = "text/css">
      .regText {font: 16pt 'Times New Roman'}
      .wordText {color: blue;}
    </style>
  </head>
  <body>
    <p class = "regText">
      The state of
      <span class = "wordText";
         onmouseover = "this.style.color = 'red';
                        this.style.fontStyle = 'italic';
                        this.style.fontSize = '24pt';"
         onmouseout = "this.style.color = 'blue';
                       this.style.fontStyle = 'normal';
                       this.style.fontSize = '16pt';">
         Washington
      </span>
      produces many of our nation's apples.
    </p>
  </body>
</html>
```

Notice that the event handlers in this example are embedded in the markup. This is one of those cases where the small amount of JavaScript needed does not justify putting it in a separate file.

Figures 6.6 and 6.7 show browser displays of the `dynFont.html` document with the mouse cursor not over, and then over, the link.

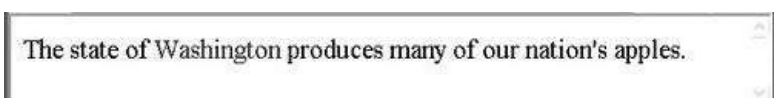

Figure 6.6 Display of `dynFont.html` with the mouse cursor not over the word

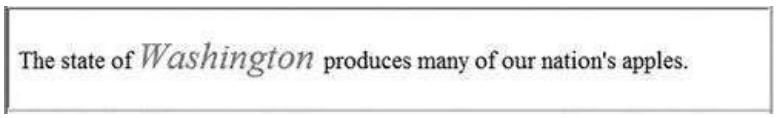

Figure 6.7 Display of `dynFont.html` with the mouse cursor over the word

6.6 Dynamic Content

We have explored the options of dynamically changing the positions of elements, their visibility, and the colors, background colors, and styles of text fonts. This section investigates changing the content of XHTML elements. The content of an element is accessed through the `value` property of its associated Java-Script object. So, changing the content of an element is not essentially different from changing other properties of the element. We now develop an example that illustrates one use of dynamic content.

Assistance to a browser user filling out a form can be provided with an associated text area, often called a *help box*. The content of the help box can change, depending on the placement of the mouse cursor. When the cursor is placed over a particular input field, the help box can display advice on how the field is to be filled in. When the cursor is moved away from an input field, the help box content can be changed to simply indicate that assistance is available.

In the next example, an array of messages that can be displayed in the help box is defined in JavaScript. When the mouse cursor is placed over an input field, the `mouseover` event is used to call a handler function that changes the help box content to the appropriate value (the one associated with the input field). The appropriate value is specified with a parameter sent to the handler function. The `mouseout` event is used to trigger the change of the content of the help box back to the "standard" value. Following is the markup document and associated JavaScript file:

```
<?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">

<!-- dynValue.html
     Uses dynValue.js
     Illustrates dynamic values
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Dynamic values </title>
    <script type = "text/javascript"  src = "dynValue.js" >
    </script>
  </head>
  <body>
    <form action = "">
      <p style = "font-weight: bold">
        <span style = "font-style: italic">
          Customer information
        </span>
        <br /><br />
        <label>
          Name:
          <input type = "text"  onmouseover = "messages(0)"
                 onmouseout = "messages(4)" />
        </label>
        <br />
        <label>
          Email:
          <input type = "text"  onmouseover = "messages(1)"
                 onmouseout = "messages(4)" />


        </label>
        <br /> <br />
        <span style = "font-style: italic">
          To create an account, provide the following:
        </span>
        <br /> <br />
        <label>
          User ID:
          <input type = "text"  onmouseover = "messages(2)"
```

```
                onmouseout = "messages(4)" />
        </label>
        <br />
        <label>
          Password:
          <input type = "password"
                 onmouseover = "messages(3)"
                 onmouseout = "messages(4)" />
        </label>
        <br />
        <textarea id = "adviceBox"  rows = "3"  cols = "50"
                  style = "position: absolute; left: 250px;
                  top: 0px">
This box provides advice on filling out the form
on this page. Put the mouse cursor over any input
field to get advice.
        </textarea>
        <br /><br />
        <input type = "submit"  value = "Submit" />
        <input type = "reset"  value = "Reset" />
      </p>
    </form>
  </body>
</html>
```

```
// dynValue.js
//   Illustrates dynamic values

var helpers = ["Your name must be in the form: \n \
 first name, middle initial., last name",
  "Your email address must have the form: \
 user@domain",
  "Your user ID must have at least six characters",
  "Your password must have at least six \
 characters and it must include one digit",
  "This box provides advice on filling out\
 the form on this page. Put the mouse cursor over any \
 input field to get advice"]

// ********************************************************
// The event handler function to change the value of the
// textarea

function messages(adviceNumber) {
  document.getElementById("adviceBox").value =
                                  helpers[adviceNumber];
}
```

Note that the backslash characters that terminate some of the lines of the literal array of messages specify that the string literal is continued on the next line. Figure 6.8 shows a browser display of the document defined in `dynValue.html` when the mouse cursor is over the *User ID* input field.

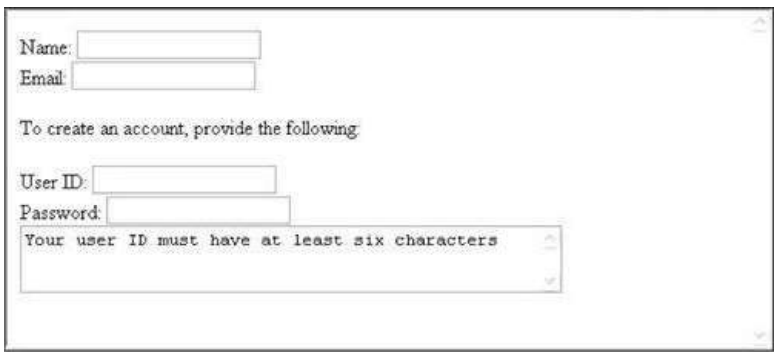

Figure 6.8 Display of `dynValue.html` with the cursor over *User ID*

6.7 Stacking Elements

The top and left properties allow the placement of an element anywhere in the two dimensions of the display of a document. Although the display is restricted to two physical dimensions, the effect of a third dimension is possible through the simple concept of stacked elements, such as that used to stack windows in graphical user interfaces. Although multiple elements can occupy the same space in the document, one is considered to be on top and is displayed. The top element hides the parts of the lower elements on which it is superimposed. The placement of elements in this third dimension is controlled by the z-index attribute of the element. An element whose z-index is greater than that of an element in the same space will be displayed over the other element, effectively hiding the element with the smaller z-index value. The JavaScript style property associated with the z-index attribute is zIndex.

In the next example, three images are placed on the display so that they overlap. In the XHTML description of this situation, each image tag includes an onclick attribute, which is used to trigger the execution of a JavaScript handler function. First the function defines DOM addresses for the last top element and the new top element. Then the function sets the zIndex value of the two elements so that the old top element has a value of 0 and the new top element has the value 10, effectively putting it at the top. The script keeps track of which image is currently on top with the global variable top, which is changed every time a new element is moved to the top with the toTop function. Note that the zIndex value, as is the case with other properties, is a string. This document, called stacking.html, and the associated JavaScript file are as follows:

```
<?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">

<!-- stacking.html
     Uses stacking.js
     Illustrates dynamic stacking of images
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Dynamic stacking of images </title>
    <script type = "text/javascript"  src = "stacking.js" >
    </script>
    <style type = "text/css">
      .plane1 {position: absolute; top: 0; left: 0;
               z-index: 0;}
      .plane2 {position: absolute; top: 50px; left: 110px;
               z-index: 0;}
      .plane3 {position: absolute; top: 100px; left: 220px;
               z-index: 0;}
    </style>
  </head>
  <body>
    <p>
      <img class = "plane1"  id = "C172"
           src = "../images/c172.gif"
           alt = "(Picture of a C172)"
           onclick = "toTop('C172')" />
      <img class = "plane2"  id = "cix"
           src = "../images/cix.gif"
           alt = "(Picture of a Citation airplane)"
           onclick = "toTop('cix')" />
      <img class = "plane3"  id = "C182"
           src = "../images/c182.gif"
           alt = "(Picture of a C182)"
           onclick = "toTop('C182')" />
    </p>
  </body>
</html>
```

```
// stacking.js
//   Illustrates dynamic stacking of images
var top = "C172";

// The event handler function to move the given element
// to the top of the display stack
function toTop(newTop) {

// Set the two dom addresses, one for the old top
// element and one for the new top element
  domTop = document.getElementById(top).style;
  domNew = document.getElementById(newTop).style;

// Set the zIndex properties of the two elements, and
// reset top to the new top
  domTop.zIndex = "0";
  domNew.zIndex = "10";
  top = newTop;
}
```

Figures 6.9, 6.10, and 6.11 show the document described by stacking.html in three of its possible configurations.

Figure 6.9 The initial display of `stacking.html` (photographs courtesy of Cessna Aircraft Company)

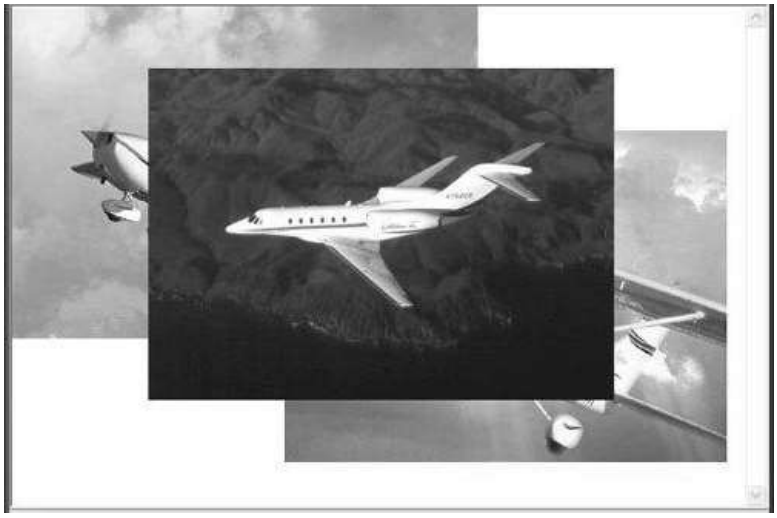

Figure 6.10 The display of `stacking.html` after clicking the second image (photographs courtesy of Cessna Aircraft Company)

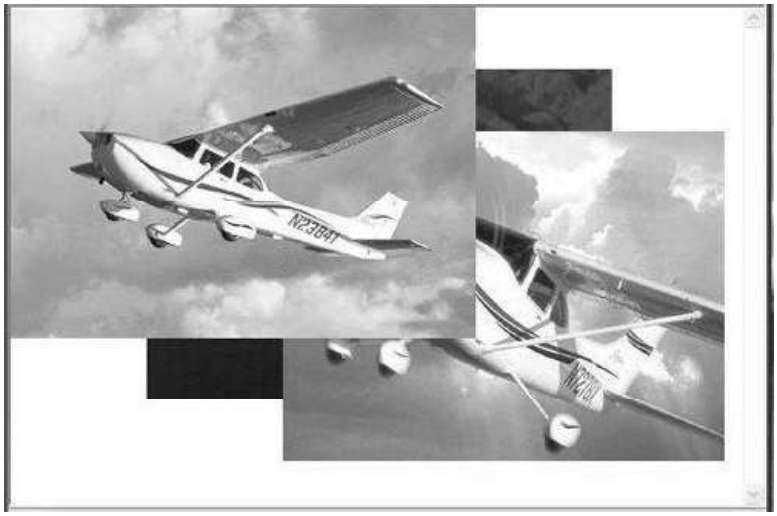

Figure 6.11 The display of `stacking.html` after clicking the bottom image (photographs courtesy of Cessna Aircraft Company)

6.8 Locating the Mouse Cursor

Recall from Chapter 5 that every event that occurs while an XHTML document is being displayed creates an event object. This object includes some information about the event. A mouse-click event is an implementation of the `MouseEvent` interface, which defines two pairs of properties that provide geometric coordinates of the position of the element in the display that created the event. One of these pairs, `clientX` and `clientY`, gives the coordinates of the element relative to the upper left corner of the browser display window, in pixels. The other pair, `screenX` and `screenY`, also gives coordinates of the element, but relative to the client

upper-left corner of the browser display window, in pixels. The other pair, `screenX` and `screenY`, also gives coordinates of the element, but relative to the client computer's screen. Obviously, the former pair is usually more useful than the latter.

In the next example, `where.html`, two pairs of text boxes are used to display these four properties every time the mouse button is clicked. The handler is triggered by the `onclick` attribute of the body element. An image is displayed just below the display of the coordinates, but only to make the screen more interesting.

The call to the handler in this example sends `event`, which is a reference to the event object just created in the element, as a parameter. This is a bit of magic, because the event object is implicitly created. In the handler, the formal parameter is used to access the properties of the coordinates. Note that the handling of the event object is not implemented the same way in the popular browsers. The Firefox browsers send it as a parameter to event handlers, whereas Microsoft browsers make it available as a global property. The code in `where.html` works for both of these approaches by sending the event object in the call to the handler. It is available in the call with Microsoft browsers because it is visible there as a global variable. Of course, for a Microsoft browser, it need not be sent at all. The `where.html` document and it's associated JavaScript file are as follows:

```
<?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">

<!-- where.html
     Uses where.js
     Illustrates x- and y-coordinates of the mouse cursor
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Where is the cursor? </title>
    <script type = "text/javascript"  src = "where.js" >
    </script>
  </head>
  <body onclick = "findIt(event)">
    <form action = "">
      <p>
        Within the client area: <br />
        x:
        <input type = "text"  id = "xcoor1"  size = "4" />
        y:
        <input type = "text"  id = "ycoor1"  size = "4" />
        <br /><br />
        Relative to the origin of the screen coordinate system:
        <br />
        x:
        <input type = "text"  id = "xcoor2"  size = "4" />
        y:
        <input type = "text"  id = "ycoor2"  size = "4" />
      </p>
    </form>
    <p>
      <img src = "../images/c172.gif"  alt = "(Picture of C172)" />
    </p>
  </body>
</html>
```

```
// where.js
//   Show the coordinates of the mouse cursor position
//   in an image and anywhere on the screen when the mouse
//   is clicked

// The event handler function to get and display the
//  coordinates of the cursor, both in an element and
//  on the screen
function findIt(evt) {
  document.getElementById("xcoor1").value = evt.clientX;
  document.getElementById("ycoor1").value = evt.clientY;
  document.getElementById("xcoor2").value = evt.screenX;
  document.getElementById("ycoor2").value = evt.screenY;
}
```

Figure 6.12 shows a browser display of `where.html`.

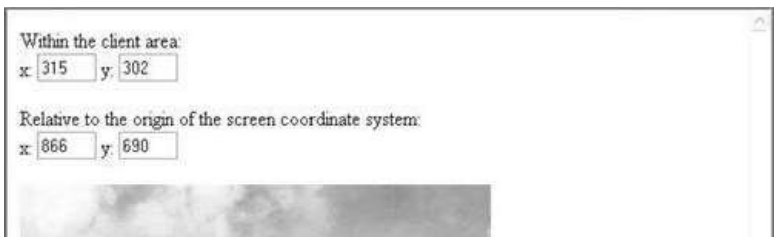

Figure 6.12 Display of `where.html` (the cursor was in the tail section of the plane)

One interesting note about the preceding cursor-finding example is that, with IE8, the mouse clicks are ignored if the mouse cursor is below the last element on the display. The FX3 browser always responds the same way, regardless of where the cursor is on the display.

6.9 Reacting to a Mouse Click

The next example is another one related to reacting to mouse clicks. In this case, the `mousedown` and `mouseup` events are used, respectively, to show and hide the message "Please don't click here!" on the display under the mouse cursor whenever the mouse button is clicked, regardless of where the cursor is at the time. The offsets (`-130` for `left` and `-25` for `top`) modify the actual cursor position so that the message is approximately centered over it. Here is the document and it's associated JavaScript file:

```
<?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">

<!-- anywhere.html
     Uses anywhere.js
     Display a message when the mouse button is pressed,
     no matter where it is on the screen
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Sense events anywhere </title>
    <script type = "text/javascript"  src = "anywhere.js" >
    </script>
  </head>
  <body onmousedown = "displayIt(event);"
        onmouseup = "hideIt();">
    <p>
      <span id= "message"
            style = "color: red; visibility: hidden;
                     position: relative;
                     font-size: 20pt; font-style: italic;
                     font-weight: bold;">
        Please don't click here!
      </span>
      <br /><br /><br /><br /><br /><br /><br /><br />
      <br /><br /><br /><br /><br /><br /><br /><br />
    </p>
  </body>
</html>


// anywhere.js
//   Display a message when the mouse button is pressed,
//   no matter where it is on the screen

// The event handler function to display the message
function displayIt(evt) {
  var dom = document.getElementById("message");
  dom.style.left = (evt.clientX - 130) + "px";
  dom.style.top = (evt.clientY - 25) + "px";
  dom.style.visibility = "visible";
}

// ------------------------------------------------------------
```

```
// ***************************************************
// The event handler function to hide the message
function hideIt() {
  document.getElementById("message").style.visibility =
      "hidden";
}
```

6.10 Slow Movement of Elements

So far, only element movements that happen instantly have been considered. These movements are controlled by changing the `top` and `left` properties of the element to be moved. The only way to move an element slowly is to move it by small amounts many times, with the moves separated by small amounts of time. JavaScript has two `Window` methods that are capable of this task: `setTimeout` and `setInterval`.

The `setTimeout` method takes two parameters: a string of JavaScript code to be executed and a number of milliseconds of delay before executing the given code. For example, the call

```
setTimeout("mover()", 20);
```

causes a 20-millisecond delay, after which the function `mover` is called.

The `setInterval` method has two forms. One form takes two parameters, exactly as does `setTimeout`. It executes the given code repeatedly, using the second parameter as the interval, in milliseconds, between executions. The second form of `setInterval` takes a variable number of parameters. The first parameter is the name of a function to be called, the second is the interval in milliseconds between the calls to the function, and the remaining parameters are used as actual parameters to the function being called.

The example presented here, `moveText.html`, moves a string of text from one position (100, 100) to a new position (300, 300). The move is accomplished by using `setTimeout` to call a mover function every millisecond until the final position (300, 300) is reached. The initial position of the text is set in the span element that specifies the text. The `onload` attribute of the body element is used to call a function, `initText`, to initialize the *x*- and *y*-coordinates of the initial position to the `left` and `top` properties of the element and call the mover function.

The mover function, named `moveText`, takes the current coordinates of the text as parameters, moves them one pixel toward the final position, and then, using `setTimeout`, calls itself with the new coordinates. The recomputation of the coordinates is complicated by the fact that we want the code to work regardless of the direction of the move.

One consideration with this script is that the properties of the coordinates are stored as strings with units attached. For example, if the initial position of an element is (100, 100), its `left` and `top` property values both have the string value `"100px"`. To change the properties arithmetically, they must be numbers. Therefore, the property values are converted to strings with just numeric digit characters in the `initText` function by stripping the nondigit unit parts. This conversion allows them to be coerced to numbers when they are used as operands in arithmetic expressions. Before the `left` and `top` properties are set to the new coordinates, the units abbreviation (in this case, `"px"`) is catenated back onto the coordinates.

It is interesting that, in this example, placing the event handler in a separate file avoids a problem that would occur if the JavaScript were embedded in the markup. The problem is the use of XHTML comments to hide JavaScript and having possible parts of XHTML comments embedded in the JavaScript. For example, if the JavaScript statement `x--;` is embedded in an XHTML comment, the validator complains that the `--` in the statement is an invalid comment declaration.[3]

In the code file, `moveTextfuns.js`, note the complexity of the call to the `moveText` function in the call to `setTimeout`. This level of complexity is required because the call to `moveText` must be built from static strings with the values of the variables `x` and `y` catenated in.

The `moveText.html` document and the associated JavaScript file, `moveTextfuns.js`, are as follows:

```
<?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">

<!-- moveText.html
     Uses moveTextfuns.js
     Illustrates a moving text element
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Moving text </title>
     <script type = "text/javascript"
             src = "moveTextfuns.js">
     </script>
  </head>
<!-- Call the initializing function on load, giving the
      destination coordinates for the text to be moved
      -->
  <body onload = "initText()">

<!-- The text to be moved, including its initial position -->
    <p>
      <span id = 'theText' style =
                "position: absolute; left: 100px; top: 100px;
                 font: bold 20pt 'Times Roman';
                 color: blue;"> Jump in the lake!
```