Sams Teach Yourself CSS in 24 Hours

by Kynn Bartlett

ISBN: 0-672-32906-9

www.samspublishing.com

BONUS CONTENT

BONUS HOUR 1: **CSS and JavaScript Website**

- What Is JavaScript?
- Using JavaScript with CSS

BONUS HOUR 2: **CSS and XML Website**

- What Is XML?
- Displaying XML
- XML-based Languages and CSS

BONUS WEB HOUR 1

CSS and JavaScript

What You'll Learn in This Hour:

- The basic concepts of JavaScript
- How to use HTML events to trigger JavaScripts in web pages
- How CSS can be used with JavaScript to produce Dynamic HTML
- How to change the appearance of a page based on JavaScript controls
- How to detect specific browsers with JavaScript and provide an alternate style sheet for each

In addition to HTML and Cascading Style Sheets, JavaScript is one of the primary languages spoken by web browsers. The HTML code provides the structure of the content, the style sheet provides the presentation, and the JavaScript provides the interactivity and actions. JavaScript and CSS go hand in hand in building the user's experience with the website.

What Is JavaScript?

JavaScript is a programming language originally created by Netscape but now used by nearly all web browsers, including Firefox, Safari, Opera, and Internet Explorer.

By the Way

JavaScript actually goes by several names. The original name was LiveScript, but they changed it to JavaScript because Java was a hot buzzword at the time. (The less cynical reason is that the syntax of JavaScript is very loosely based on the Java programming language.) The standardized version of JavaScript is called ECMAScript and is published by ECMA, the European Association for Standardizing Information and Communication Systems (http://www.ecma-international.org/). Microsoft uses the term JScript to refer to the specific type of JavaScript supported by Internet Explorer.

JavaScript has something in common with Cascading Style Sheets; each browser has quirks in how it understands JavaScript. Just as a CSS designer has to be aware of browser support issues, so does a JavaScript programmer need to take special care to account for browser limitations.

In this hour, there's not enough time to cover everything you need to know about JavaScript; that could take a whole book itself. Instead, I'll tell you about what JavaScript can do, and I hope to whet your appetite for learning more if you haven't worked with it before. If you already know JavaScript, this hour will show you how to use it with CSS styles. Some example scripts in this hour will show you specific cases of JavaScript and style sheets working together to produce the end result.

Did you Know?

In fact, there are several good books about JavaScript. If you like the Sams Teach Yourself approach to learning, you'll want to pick up *Sams Teach Yourself JavaScript in 24 Hours, Fourth Edition*; ISBN: 0672328798, by Michael G. Moncur.

JavaScript runs on the *browser side*, which means that it doesn't run on a web server but within a web browser. Scripts can be written that produce a wide variety of effects, from validating form input to creating animation. The ones we are most concerned with in this hour are those that interact with CSS—setting styles, hiding or displaying content, or positioning elements on the screen.

As a programming language, JavaScript has a different feel than either CSS or HTML. A script is a set of instructions that are executed by the browser in a specific order, rather than a set of style rules or marked-up content. Execution means that the browser follows the instruction, and then the browser goes on to the next instruction in the script. In general, a script starts running when the browser encounters it on the page. Similar to CSS, JavaScript instructions are separated by semi-colons, and distinct sections are contained in matching curly braces.

Did you Know?

What about other scripting languages? Why would you want to use JavaScript instead of another language? In short, JavaScript is the only browser-side language that's well supported by all major browsers. Many other web programming languages, such as Perl, ASP, PHP, or Cold Fusion, run on the web server rather than on the browser, and they may require your web host to install specific software on the server. Server-side programming languages can be used with CSS because they produce normal HTML files, no matter how that code is generated by server code.

Scripts that execute within the browser are written in client-side languages, such as JavaScript, and don't require anything special on the back end. Other examples of languages that execute on the client side include VBScript and Java. VBScript

functions only on Internet Explorer, whereas Java is many times more complex than JavaScript. For these reasons, JavaScript is the language of choice for client-side programming on the Web.

A script contains constructs found in programming languages, such as functions, variables, and conditional statements. If you haven't programmed before, these might not be familiar concepts. A *function* is a part of the script that doesn't immediately execute when the browser encounters it but instead defines a section of code that can be executed upon request elsewhere in the script. A *variable* is a named "storage bin" that holds a value that may be accessed within the script. A *conditional statement* creates a branch in the execution order; if a given condition exists, one set of instructions is executed, and if not, either nothing is executed or another set of instructions is executed.

To use JavaScript with HTML, you use the `<script>` tag. The `<script>` tag can be placed in either the `<head>` section or the `<body>` section of the web page. An example of JavaScript in HTML is shown in Listing web1.1, which is a very trivial number quiz.

LISTING WEB1.1 A Simple Interactive Page Created with JavaScript

```
<!-- demo-w1.1.html -->
<html>
  <head>
    <title>JavaScript Demo</title>
    <script type="text/javascript" language="JavaScript">
      var magicNumber = Math.floor(Math.random()*100+0.5);
      var isOdd = magicNumber % 2;
      function pressOdd() {
        if (isOdd) { window.alert("Correct!\n\n" +
                     magicNumber + " is odd."); }
           else    { window.alert("Try again!"); } }
      function pressEven() {
        if (isOdd) { window.alert("Try again!"); }
           else    { window.alert("Correct!\n\n" +
                     magicNumber + " is even."); } }
    </script>
  </head>
  <body>
    <h1>The Great Odd/Even Game!</h1>
    <script type="text/javascript" language="JavaScript">
      document.write("<h2>The magic number is..." +
                     magicNumber + "</h2>");
    </script>
    <p>Is this odd or even? What's your guess?</p>
    <form> <button onclick="pressOdd()">Odd</button>
           <button onclick="pressEven()">Even</button>
           <button onclick="document.location.reload()"
                   >Play Again!</button></form>
  </body>
</html>
```

There are two script sections in Listing web1.1. The first is within the `<head>` element of the page, and when the page loads, that script randomly picks a number, assigns it to the `magicNumber` variable, determines whether it is odd or even and assigns the result of that determination to the `isOdd` variable, and finally sets up two functions, `pressOdd()` and `pressEven()`. Those functions don't do anything until they are triggered elsewhere in the page.

The second script, in the `<body>` of the page, creates HTML by using the built-in `document.write()` function to add an `<h2>` tag and show the value of the `magicNumber` variable.

The `<button>` elements are used to create form buttons that display "odd," "even," or "choose another." The `onclick` attribute on the first `<button>` triggers the `pressOdd()` function; the `onclick` attribute on the second `<button>` triggers the `pressEven()` function. The third button has an `onclick` attribute that reloads the current location. These `onclick` attributes are HTML intrinsic events, and you'll learn more about them later this hour.

When the `pressOdd()` or `pressEven()` functions are triggered by button clicks, the browser creates a pop-up alert window. The contents of this window depend on whether the correct answer was given, as determined by the value of the `isOdd` variable. In Figure web1.1, you can see an example of this script in action.

FIGURE WEB1.1 This game gets boring very quickly.

HTML Events

A script normally is executed when the browser encounters it; in other words, when the page is being displayed and the browser comes across the script within a `<script>` tag, it executes. However, you can also set code to execute upon the fulfillment of certain conditions, usually the result of the user's actions. These are called *intrinsic events*. Intrinsic events are defined by the HTML specification and are attributes that can be set on HTML tags. For example, in Listing web1.1, you saw the `onclick` attribute used as follows:

```
<button onclick="pressOdd()">Odd</button>
<button onclick="pressEven()">Even</button>
<button onclick="document.location.reload()"
        >Play Again</button></form>
```

The value assigned to an intrinsic event attribute is a small snippet of JavaScript. In the first two examples, the `onclick` attribute calls a function that is defined earlier in the page. In the third example, the `onclick` attribute calls a built-in method, `document.location.reload()`, which reloads the current document. These events happen only when the button is clicked.

A complete list of HTML intrinsic event attributes, and the actions that trigger those events, is shown in Table web1.1.

TABLE WEB1.1 Intrinsic Events in HTML

Attribute	Triggering Event
`onBlur`	When the element loses focus (links and form controls only)
`onChange`	When the current selection changes (`<select>`, `<input>`, and `<textarea>` tags only)
`onClick`	When the mouse clicks over an element
`onDblClick`	When the mouse is double-clicked over an element
`onFocus`	When the element receives focus (links and form controls only)
`onKeyDown`	When a key is pressed down
`onKeyPress`	When a key is pressed and released
`onKeyUp`	When a pressed key is released
`onLoad`	When the page loads (`<body>` and `<frameset>` tags only)
`onMouseDown`	When the mouse clicks down over an element

TABLE WEB1.1 Continued

Attribute	Triggering Event
onMouseMove	When the mouse pointer moves while over an element
onMouseOut	When the mouse pointer moves off an element
onMouseOver	When the mouse pointer moves over an element
onMouseUp	When the mouse button is released over an element
onSelect	When the user selects text in a text field (<input> and <textarea> tags only)
onSubmit	When a form is submitted (<form> tags only)
onUnLoad	When the page "unloads" (<body> and <frameset> tags only)

For readability's sake, I have capitalized the first letter of words embedded within event attribute names. Because HTML is not case sensitive, it doesn't matter whether you write onMouseMove or onmousemove.

Watch Out!

However, if you are using XHTML rather than HTML, it does matter! XHTML is case sensitive and requires all attributes to be written in lowercase. So for an XHTML document, you could write only onmousemove, not onMouseMove.

Some of these events should seem familiar to you. The onMouseOver event is similar to the :hover pseudo-class of CSS, and onFocus is like :focus. However, each of these events is actually just half of the action; when you move the mouse to hover over an element, two things happen. First, it's now "over" that element, and second, it's no longer where it was before. This is an onMouseOver event for the new element, and an onMouseOut event for the old location.

How JavaScript Views a Document

In a CSS context, an HTML file consists of a series of nested boxes that have styles applied to them. In JavaScript, HTML elements are referenced as objects because JavaScript is an object-based language. Each object in JavaScript can have associated methods or properties that can be accessed by scripts and can also have other objects under them within a hierarchy. A method is like a predefined function, and a property is a value.

To use a method, property, or subobject, you need to indicate what object you're talking about. You do so by putting the object name before the name of the method, property, or subobject and separating the two with a period (`.`). Here is an example from earlier this hour: the function to reload the current document:

```
document.location.reload()
```

This says to use the `reload()` method, which is associated with the `location` object, which is part of the `document` object. You can think of this as somewhat similar to the way file locations are indicated on a computer's hard drive, such as `C:\My Documents\Word Files\Hour 23.doc`, or even the way websites are organized with a file hierarchy.

To access the properties of a web document from within a script, you use a method called the *Document Object Model* (DOM). This is a standard way of accessing HTML elements, content, and attributes that was developed by the World Wide Web Consortium. Unfortunately, many browsers have their own ways of accessing HTML content from within JavaScript, and most of them fall short of measuring up to the standards of the W3C's DOM. This is one of the reasons that cross-browser HTML can be difficult.

Dynamic HTML

Dynamic HTML is a term you may have heard before, and depending on who is using the term, it means different things. In most cases, it means more than just "exciting web design" and commonly refers to web pages that move and change interactively, based on the user's actions, without going back to the web server to fetch a new web page.

In simple terms, dynamic HTML is the interaction of HTML, JavaScript, and CSS properties to produce those interactive effects. An example of dynamic HTML would be a complex mouseover that displays additional information as you move the mouse on the screen, or an animated banner that rolls out to show a list of options from which to choose. These are all the result of combining JavaScript's actions with CSS's presentation.

Using JavaScript with CSS

JavaScript was designed so that it works well with Cascading Style Sheets; a script can set or unset CSS properties. You accomplish this by using the `style` object, part of the DOM that provides access to any HTML object's properties.

The `style` object has properties that generally correspond to CSS properties, although each browser supports a variable set. Most of the basic properties, such as `color`, `background`, and `font`, are well supported. For example, to set the `color` of an object using JavaScript, you'd do the following:

```
object.style.color = blue;
```

Did you Know?

> CSS properties that are two (or more) words separated by hyphens, such as `background-image`, are written in JavaScript as the first word in lowercase, no hyphen, and then the second word with an uppercase letter—for example, `backgroundImage`.

There are several ways in JavaScript to identify the exact object you want to work with; the easiest is to employ the `document.getElementById()` method. This selects the object corresponding to a given `id` value, so you can use it with only HTML tags that have `id` attributes. For example, if you want to change the color of an `<h1>` tag with an `id` of `main`, you can use the following line in your JavaScript:

```
document.getElementById("main").style.color = blue;
```

This isn't the only way to use JavaScript to manipulate styles, but it is both simple and powerful. Several of the examples that follow use this straightforward method to set CSS property values. All the following examples can be downloaded here on the book's website.

JavaScript and Dynamic Styles

The first example of using JavaScript with CSS can be seen in Listing web1.2, which allows the user to select a theme, and then uses that theme to display text. A theme is simply a collection of styles.

LISTING WEB1.2 A Theme Picker in JavaScript

```
<!-- picker-w1.2.html -->
<html>
  <head>
    <title>Theme Picker</title>
    <style type="text/css">
      body { padding: 1em;
             font-family: Verdana, sans-serif; }
      #eg { width: 30em; min-height: 6em;
            max-width: 700px;
            border: 2px solid black; padding: 1em;
            margin: 2em auto 0px auto; }
    </style>
    <script type="text/javascript" language="JavaScript">
```

LISTING WEB1.2 Continued

```
    function chooseOne() {
      eg = document.getElementById("eg");
      i = document.pick.chooser.selectedIndex;
      newTheme = document.pick.chooser.options[i].value;
      switch(newTheme) {
        case "default" :
          eg.style.color = "black";
          eg.style.background = "white";
          eg.style.font = "1em serif";
          break;
        case "inverse" :
          eg.style.color = "white";
          eg.style.background = "black";
          eg.style.font = "1em serif";
          break;
        case "typewriter" :
          eg.style.color = "black";
          eg.style.background = "white";
          eg.style.font = "1em monospace";
          break;
        case "microscriptic" :
          eg.style.color = "white";
          eg.style.background = "gray";
          eg.style.font = "0.5em cursive";
          break;
        case "huge" :
          eg.style.font = "2em Verdana";
          eg.style.color = "gray";
          eg.style.background = "white";
          break;
        case "southwest" :
          eg.style.font = "16pt Papyrus";
          eg.style.color = "white";
          eg.style.background =
                   "#CCA580 url('swbg.jpg') no-repeat";
          break; }
      }
  </script>
</head>
<body onload="chooseOne()">
  <h1>Theme Picker</h1>
  <form name="pick" id="pick">
    <label for="chooser">Choose a theme from this list, and it
      will be shown in the box below.</label>
    <select name="chooser" id="chooser" onchange="chooseOne()">
      <option value="default">Default</option>
      <option value="inverse">Inverse</option>
      <option value="typewriter">Typewriter</option>
      <option value="microscriptic">Microscriptic</option>
      <option value="huge">HUGE</option>
      <option value="southwest">Southwest</option>
    </select>
  </form>
  <div id="eg">
    <h2>Darned Foxes</h2>
    <p>The quick brown fox jumped over the lazy dog. The
```

LISTING WEB1.2 Continued

```
        little dog laughed to see such a sight.</p>
    </div>
  </body>
</html>
```

This JavaScript uses the `<select>` tag to create a pull-down menu. When a new selection is made, the `onchange` attribute of the `<select>` tag triggers the `chooseOne()` function. The current value of the pull-down menu is read in these lines, which use the DOM to access the value:

```
i = document.pick.chooser.selectedIndex;
newTheme = document.pick.chooser.options[i].value;
```

That value is then used in a `switch`, a conditional statement that chooses among several options based on the specified value. Each of the `case` sections within the `switch` sets a different group of `style` values, determined by the chosen theme. These styles are set on the `<div>` with the `id` of `eg` (for example). You can see this in action in Figure web1.2 or by downloading the file from here on the book's website.

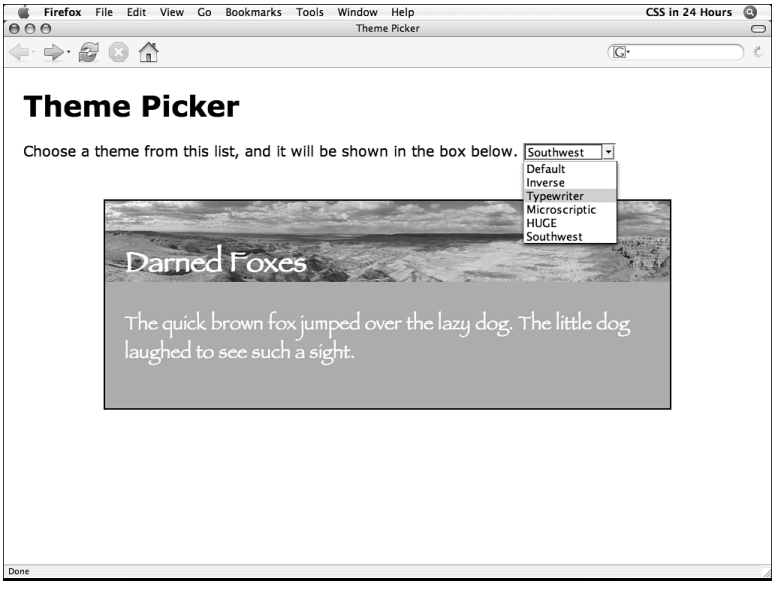

FIGURE WEB1.2 Choosing a theme changes the styling of the demo box dynamically.

JavaScript and Visibility

In Hour 16, "Borders and Boxes," you learned about the `visibility` property, which lets you designate certain CSS boxes as visible or hidden. This property isn't all that useful by itself, but when combined with JavaScript it comes into its own.

JavaScript enables you to turn off and on the visibility of a display box so you can hide or show parts of the page dynamically.

Listing web1.3 is an example of visibility under JavaScript control.

LISTING WEB1.3 An HTML Page That Uses JavaScript to Display Different Panels

```
<!-- dogs-w1.3.html -->
<html>
  <head>
    <title>Dynamic Dogs</title>
    <style type="text/css">
      body { font-family: Verdana, sans-serif;
             padding: 2em; }
      #menu { margin: 1em; text-align: center; }
      #menu h1 { display: inline; }
      #menu a:link, #menu a:visited {
              margin-left: 2em; font-size: 2em;
              text-decoration: none;
              border-bottom: dotted 1px gray; }
      #data { position: relative; margin: 0 auto;
              width: 20em; height: 20em;
              background-color: gray; }
      #data div { position: absolute; top: 0.5em; left: 0.5em;
                  width: 18em; height: 18em;
                  visibility: hidden; overflow: hidden;
                  background-color: white; padding: 0.5em; }
      #data #blank { visibility: visible;
                     background-color: silver; }
      #blank:before { content:
          "Mouseover (or tab to) each name to read about our dogs.";
         }
      .dogpic { float: left; margin: 0 1em; }
    </style>
    <script type="text/javascript" language="JavaScript">
      function showDog(dog) {
        var dogElement = document.getElementById(dog);
        dogElement.style.visibility = "visible";
        document.getElementById("blank") = "hidden"; }
      function hideDog(dog) {
        var dogElement = document.getElementById(dog);
        dogElement.style.visibility = "hidden";
        document.getElementById("blank") = "visible"; }
    </script>
  </head>
  <body>
    <div id="menu">
      <h1>Our Dogs:</h1>
      <a href="#kim"
         onmouseover="showDog('kim')"
         onfocus="showDog('kim')"
         onmouseout="hideDog('kim')"
         onblur="hideDog('kim')">Kim</a>
      <a href="#angie"
         onmouseover="showDog('angie')"
         onmouseout="hideDog('angie')"
```

LISTING WEB1.3 Continued

```
        onfocus="showDog('angie')"
        onblur="hideDog('angie')">Angie</a>
      <a href="#nying"
        onmouseover="showDog('nying')"
        onmouseout="hideDog('nying')"
        onfocus="showDog('nying')"
        onblur="hideDog('nying')">Nying</a>
    </div>
    <div id="data">
      <div id="blank"></div>
      <div id="kim">
        <h2>Kim</h2>
        <img src="kim.jpg" alt="[Kim Picture]" class="dogpic">
        <p>Kim was the largest of our dogs and was very cute.
          He was lazy but lovable, and loved to eat.</p></div>
      <div id="angie">
        <h2>Angie</h2>
        <img src="angie.jpg" alt="[Angie Picture]" class="dogpic">
        <p>Angie was born first and was very smart. She loved
          to explore and bark.</p></div>
      <div id="nying">
        <h2>Nying</h2>
        <img src="nying.jpg" alt="[Nying Picture]" class="dogpic">
        <p>Nying was the runt of the litter and was a little
          bit crazy. She loved to sleep under Kynn's desk.</p></div>
    </div>
  </body>
</html>
```

This page consists of an embedded style sheet with positioning CSS and generated content, an embedded script, multiple event triggers, and overlapping `<div>` elements, so it's somewhat complex.

If you look carefully, you'll notice that the `<div>` elements within the `<div id="data">` element, including the blank `<div>`, all have the same location and dimensions specified in the style sheet. Normally these would overlap, and their order would be determined by the `z-index` property. However, they are also set to `visibility: hidden`, which means none of them appear on the page, except the `<div id="blank">` element. The blank `<div>` is empty, but content generation adds a short instruction blurb. You can see how this all looks in Figure web1.3.

When you move the mouse over one of the links on the top menu bar—for example, the name Kim—the `onmouseover` event triggers the `showDog()` function and gives it the value `kim`. That value is used to select the correct `id` and sets that `<div>` to `visibility: visible` while setting the blank `<div>` to `hidden`. This is shown in Figure web1.4.

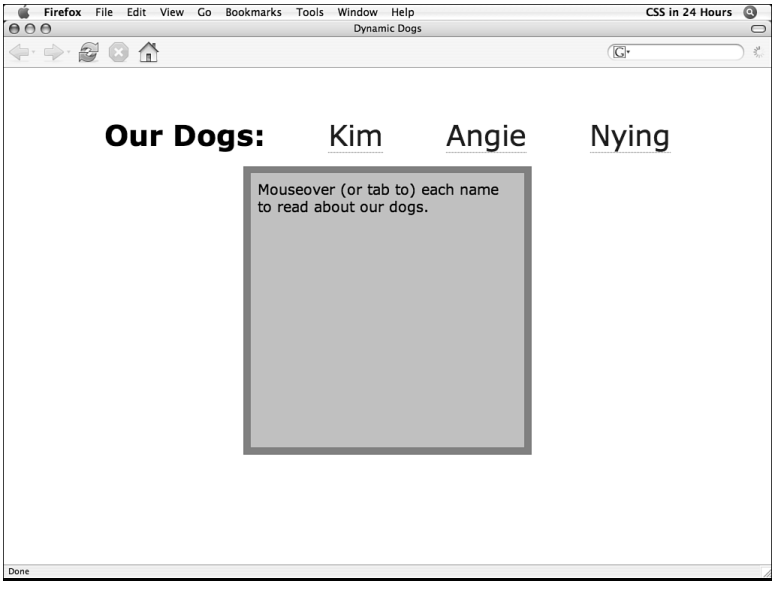

FIGURE WEB1.3 The initial list of dogs, with instructions added via content generation.

FIGURE WEB1.4 Mousing over the name *Kim* displays the information on that dog.

When the mouse is moved off the link, the `hideDog()` function is triggered by the `onmouseoff` event, and that reverses the `showDog()` function by hiding the dog and making the blank `<div>` visible once more. You can use this effect to make complex

mouseovers that progressively reveal more information, including helpful tips about a link or additional content.

JavaScript and Alternate Style Sheets

One recurring theme throughout this book has been browser irregularities. Several workarounds have been presented in Hour 24, "Troubleshooting and Browser Hacks," to deal with those deficiencies. With JavaScript, you can use a technique called *browser detection* (or sometimes browser sniffing).

Browser detection uses the `navigator` object, which returns information about the current browser, to determine which instructions to execute. Listing web1.4 is an example of such a script, which looks for Netscape 4 or earlier and, if it finds it, provides an alternate stylesheet. If not, the primary style sheet is used. This listing also demonstrates the types of information available through the navigator object, ranging from the name of the application to the platform.

LISTING WEB1.4 An HTML Page with JavaScript to Detect Netscape 4

```
<!-- alternate-w1.4.html -->
<html>
  <head>
    <title>Browser Check</title>
    <script type="text/javascript" language="JavaScript">
      document.write('<link type="text/css" rel="stylesheet" ');
      if (navigator.appName == "Netscape" &&
          parseFloat(navigator.appVersion) < 5 )
      { document.writeln(' href="n4-w1.6.css">'); }
      else
      { document.writeln(' href="main-w1.5.css">'); }
    </script>
  </head>
  <body>
    <h1>Browser Details</h1>
    <div class="section">
    <p>Your browser has provided the following information via
       the <b>navigator</b> class in JavaScript.</p>
    <script type="text/javascript" language="JavaScript">
      document.writeln("<table>");
      document.writeln("<tr><th>appCodeName:</th><td>" +
                    navigator.appCodeName + "</td></tr>");
      document.writeln("<tr><th>appName:</th><td>" +
                    navigator.appName + "</td></tr>");
      document.writeln("<tr><th>appVersion:</th><td>" +
                    navigator.appVersion + "</td></tr>");
      document.writeln("<tr><th>language:</th><td>" +
                    navigator.language + "</td></tr>");
      document.writeln("<tr><th>platform:</th><td>" +
                    navigator.platform + "</td></tr>");
      document.writeln("<tr><th>userAgent:</th><td>" +
                    navigator.userAgent + "</td></tr>");
      document.writeln("</table>");
    </script>
```

LISTING WEB1.4 Continued

```
    <div class="n4only">
      <h2>Please Upgrade!</h2>
      <p>Your browser is over 10 years old...time to download
         some new software!</p>
    </div>
    <noscript>
      <p>...I'm sorry, I couldn't determine anything about your
         browser using JavaScript.</p>
    </noscript>
    </div>
  </body>
</html>
```

The default style sheet is a gray background with white text on a black box, with a tab header written in Verdana font, as shown in Listing web1.5.

LISTING WEB1.5 The Default Style Sheet for the Page

```
/* main-w1.5.css */

body { color: white; background-color: gray;
       font-family: Verdana, sans-serif;
       padding: 2em; }

h1 { margin: 0; width: 40%;
     background-color: black;
     padding: 0 5px;
     position: relative;
     border: 2px solid white;
     border-bottom: 2px solid black;
     top: 2px;
     z-index: 5; }

.n4only { display: none; }

.section
   { border: 2px solid white;
     background-color: black;
     padding: 0 5px; }

.section:after {
     content: "Also, your browser understands"
              " generated content. Hooray!";
     padding: 0 5px; display: block; margin: 1em 0; }

th { text-align: left; vertical-align: top;
     width: 30%; }
td { vertical-align: top; color: yellow; }
```

The alternate style sheet for Netscape 4 is black text on a white background in Courier font. This style sheet is listed in Listing web1.6. These specific styles were chosen just to be visually distinct for purposes of illustration. If you were using

browser detection on a real site, you'd put styles that cause problems in Netscape 4 in `main-w1.5.css`, and you would put only those styles considered safe in Netscape 4 in the `n4-1.6.css` file.

LISTING WEB1.6 The Style Sheet Designed for Netscape 4

```
/* n4-w1.6.css */
/* Alternate stylesheet for Netscape 4 */

body, td, th {
      background-color: white; color: black;
      font-family: "Courier New", monospace; }

table { border: 2px solid black; }
tr,td { text-align: top; }
```

When a web browser loads the page, it executes the JavaScript in the `<head>` and creates a `<link>` to load the right style sheet. The rest of the page displays the properties you can access via the `navigator` object. An example of this script in action is shown in Figure web1.5; because Internet Explorer is not Netscape 4, the default style sheet is applied, and the page is gray.

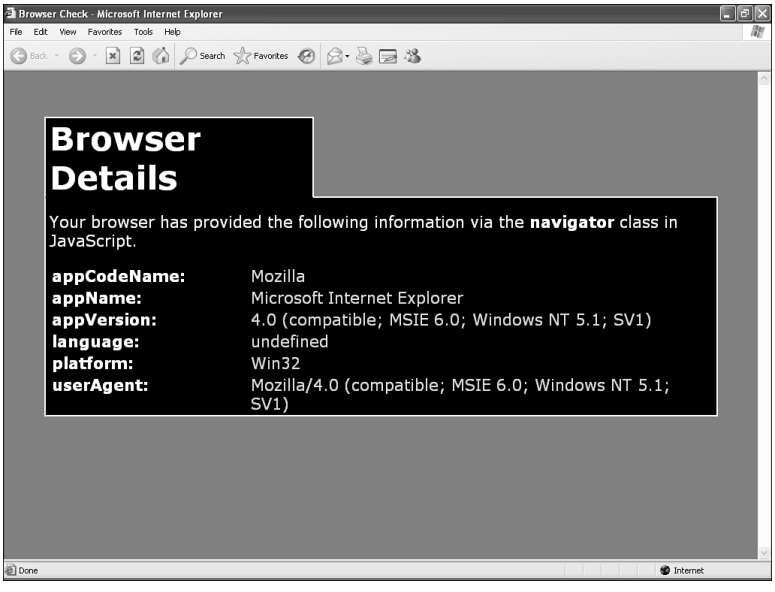

FIGURE WEB1.5 Internet Explorer 6 uses the default style sheet.

To determine whether the browser is Netscape 4, the script uses this conditional statement:

```
if (navigator.appName == "Netscape" &&
    parseFloat(navigator.appVersion) < 5 )
```

This is a compound statement—the && is read as "and"—that checks first to see whether the appName is Netscape and then uses the parseFloat() function to see whether the browser version is less than 5. In other words, it looks for Netscape 4 (or less). If you load the page in Netscape 4, you'll get the effect shown in Figure web1.6; because it's a white page with Courier text, it's using the alternate style sheet.

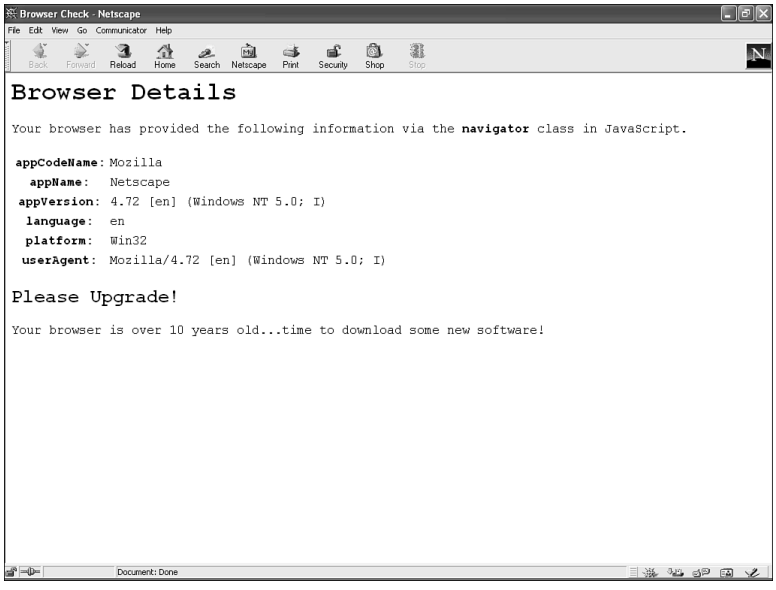

FIGURE WEB1.6
Netscape 4 receives the alternate style sheet.

Watch Out!

Browsers don't actually have to tell the truth about what they are. In fact, nearly every browser claims to be "Mozilla" in the navigator.appCodeName property, as shown in Figure web1.5. This is because web developers wrote early browser detection scripts that prevented content from being served to Internet Explorer users, so Internet Explorer masquerades as Mozilla in the appCodeName and as Netscape 4 in the appName property. The Opera browser even goes so far as to let you choose which of several other browsers it will claim to be. This means that you can't necessarily rely on browser detection. Check your scripts carefully in all browsers, just as you would with CSS styles.

Summary

JavaScript is a programming language understood by most web browsers. JavaScript adds actions to the structure of HTML and the presentation of CSS. Together, they can be used as dynamic HTML.

JavaScript can be embedded in the `<head>` or `<body>` sections of an HTML file, and it executes when the browser reaches that part of the page. JavaScript can also be set to execute when a certain condition, called an intrinsic event, happens, as determined by HTML attributes. Examples include `onClick`, `onLoad`, and `onMouseOut`.

In addition to JavaScript's other capabilities, such as validating form data and creating interactive scripts, JavaScript can be used to manipulate CSS properties. The JavaScript `style` object is used for this purpose and is part of the Document Object Model (DOM). Content can be styled dynamically, positioned, hidden, or revealed with the `style` object.

Another JavaScript object that is quite useful in conjunction with CSS is the `navigator` object. The `navigator` object provides information on the user's browser, which can then tailor the presentation to that browser's quirks in a process known as browser detection. Browser detection should be used carefully, though, as it is not always reliable.

Workshop

The workshop contains a Q&A section, quiz questions, and activities to help reinforce what you've learned in this hour. If you get stuck, the answers to the quiz can be found after the questions.

Q&A

Q. ***I can use one external style sheet on multiple pages with the*** `<link>` ***tag. Can I do the same with JavaScript?***

A. Yes. Rather than make the `<script>` tag a container, you can leave it empty and use the `src` attribute to reference a filename. Your file should end with the `.js` extension. For example:

```
<script type="text/javascript" language="JavaScript" src="runThis.js">
```

Q. ***What happens if a browser doesn't understand JavaScript?***

A. This is always a possibility. JavaScript support doesn't exist in older or limited browsers, such as Lynx, and some assistive technologies used by people with disabilities don't understand JavaScript. Also, users are able to disable JavaScript in their browser preferences. When JavaScript is not available, the `<script>` contents don't execute. Browsers instead display the contents of any `<noscript>` tags, such as the one shown in Listing web1.4.

Q. ***Is using JavaScript the only way to do browser detection?***

A. No; you can also do server-side browser detection by using a program running on the web server. This can be done with a Perl CGI script, server-side includes, ASP, PHP, or any other server programming technologies.

Quiz

1. JavaScript can be used for which of the following functions?
   - **a.** Animation
   - **b.** Validating form input
   - **c.** Choosing an alternate style sheet based on browser detection
   - **d.** Hiding parts of a page until triggered by an event
2. What is the intrinsic event that triggers when an HTML tag loses the focus?
3. Which property of the `navigator` object contains the name of the browser, unless the browser is masquerading as something else?
   - **a.** `navigator.browserName`
   - **b.** `navigator.appCodeName`
   - **c.** `navigator.appName`
   - **d.** `navigator.browser`

Answers

1. Trick question: The answer is "all of these."
2. The `onBlur` event is triggered when an element is no longer the focus.
3. The correct answer is (c), `navigator.appName`. The `navigator.appCodeName` property will almost always be `Mozilla`, and the other properties don't exist.

Exercise

Although you may not be ready to go out and start writing your own JavaScript applications after this hour, I hope this has whetted your appetite for dynamic HTML and scripting. Download and modify these scripts for your own use, or follow up by reading *Sams Teach Yourself JavaScript in 24 Hours, Fourth Edition;* ISBN: 0672328798, as your next book.

BONUS WEB HOUR 2

CSS and XML

What You'll Learn in This Hour:

- What XML is and how it is used
- How an XML-based language is defined
- How Cascading Style Sheets can be used to style an XML document
- Which styles are most useful in creating the presentation style for an XML document
- What XHTML is and how Cascading Style Sheets are used with XHTML
- How other XML-based languages, such as Scalable Vector Graphics (SVG), use CSS

Extensible Markup Language (XML) has become one of the buzzwords of the early 21st Century. Hailed as the next step in the evolution of the Web, XML promises to shake up the way we think about the design of access to information. The knowledge you've gained about using CSS with HTML will serve you well if you go on to write XML documents because CSS and XML complement each other.

What Is XML?

If you're not sure what Extensible Markup Language is, that's okay; it's been described as anything from the replacement for HTML to a universal data format, so it's no wonder that there's uncertainty. This hour won't teach you everything you need to know about XML, but it will help you understand enough to know how it works with CSS.

By the Way

To learn more about XML, check out *Sams Teach Yourself XML in 24 Hours, Complete Starter Kit, 3rd Edition;* ISBN: 067232797X, by Michael Morrison.

XML is actually not a markup language per se; instead, it's a set of rules and concepts that can be used to create markup languages. Some people call this a meta-language. Another meta-language is Simple Generalized Markup Language (SGML); SGML rules were used to build the HTML language that you're familiar with.

Using XML, you can construct any number of markup languages, either formally defined or ad hoc; all that matters is that a document follows the rules laid down by the W3C's XML specification. If one does, it's an XML document.

The primary advantage of XML is that it can be used as a lingua franca for computers and programmers. Because everything is defined in a specific format, a program that understands XML can be used with any XML-based language or document. This leads to interoperability, which is a fancy way of saying that computer applications are able to share data and output in an efficient manner.

Basic XML Concepts and Syntax

XML is purposely designed to be simple and easy to use. This is as much for the benefit of the computer applications as for the programmer because programs as well as programmers can more quickly understand a simple language with strict rules.

An XML document is written like an HTML document, with content marked up with tags denoted by less-than (<) and greater-than (>) angle brackets. Attributes, comments, and even character entities, such as `<` and `>`, are pretty much the same in XML as in HTML.

The biggest difference between XML and HTML is that XML doesn't define any specific tags that can be used for markup. If you are creating an XML document (or an XML-based language), you can make up any tag name you want. You can name a tag `<thisIsReallyImportant>` or `<Fred>` or `<yrewqjrfjasdfieh>`.

One restriction on XML documents is that they must be *well formed*. A well formed document is one where the opening and closing tags match up properly; one that is not well formed has tags that are mismatched because they're not closed in the correct order. Here is an example of mismatched tags in HTML:

```
<em>This is <a href="link.html">a link</em></a>
```

As you can see, the `<em>` is closed before the `<a>` tag, even though the `<a>` was opened more recently. Nearly any HTML web browser will display this properly, but it would still be incorrect in XML. (It's technically incorrect in HTML, too, but current browsers are forgiving.) The proper way to write this is

```
<em>This is <a href="link.html">a link</a></em>
```

XML requires that all attribute values be quoted. In HTML, you could get away with typing some values without the quotes around them, but XML is stricter in that regard.

Another important rule in XML is that all elements must have a closing tag of some sort. If a tag is empty, meaning it has no content—such as the HTML tags `<br>`, `<img>`, and `<hr>`—it is closed by a final slash written as part of the opening tag or by a closing tag, as in these examples:

```
<empty note="This element is empty" />
<empty note="This is too"></empty>
```

Both of the `<empty>` elements shown here have the same meaning; they are properly closed according to the XML rules.

XML tags are case sensitive. This means that the exact characters and case of those characters matters when you close a tag. In HTML, you can close a `<blockquote>` element with `</BLOCKQUOTE>`, `</BlockQuote>`, `</bLoCkQuOtE>`, or `</blockquote>`. In XML, the case has to match exactly—only `</blockquote>` would be valid in an XML document because the starting tag was named `<blockquote>`.

In Listing web2.1, I give you an example of a simple XML document, which describes a web accessibility tip in markup.

LISTING WEB2.1 A Sample XML Document

```
<?xml version="1.0"?>
<tippage revision="2006-07-09" xml:lang="en">
  <accesstip>
    <headline>
      Accessibility Tip: Identify Language Changes
    </headline>
    <author>
      <name>Kynn Bartlett</name>
      <email>&lt;kynn@css24.com&gt;</email>
    </author>
    <tipbody>
      <para>
        When a blind user accesses a web page with a
        screen reader, the screen reader uses a specific
        language dictionary to know how words should be
        pronounced, based on the language of the page.
        If the wrong dictionary is used, the speech
        will be very difficult to understand.
      </para>
      <para>
        If the language changes in the middle of the web
        page, you need to mark that change with the
        <code>lang</code> attribute, which can be set
```

LISTING WEB2.1 Continued

```
        on any HTML tag but is usually set on the
        <code>&lt;span&gt;</code> element. This will let
        the screen reader know which language dictionary
        to use when synthesizing speech.
      </para>
      <para paratype="note">
        The XML equivalent of the <code>lang</code>
        attribute is <code>xml:lang</code>.
      </para>
    </tipbody>
    <tipexample>
      &lt;p&gt;
        &lt;span lang="de"&gt;
          Ich bin Berliner.
        &lt;/span&gt;
        (I am a resident of Berlin)
      &lt;/p&gt;
    </tipexample>
  </accesstip>
</tippage>
```

Notice that in the listing, the `<tipexample>` element contains HTML code, but the angle brackets have been converted to character entities with `&lt;` and `&gt;`.

Also notice that this document says absolutely nothing about how to display the content; it just defines the information and leaves it at that. This is one of the primary uses of XML: completely separating presentation from content. Later this hour you'll see how CSS can be used to define that presentation.

DTDs and Schemas

To make the jump from an XML document to an XML-based language, you need to have a formal definition for a language. An XML document is not required to be part of an XML-based language, though! An XML document without a formal definition basically creates an ad hoc language as it goes along, and by the rules of XML, that's perfectly valid.

However, if you're writing an application that you mean for others to use, you may need to have the syntax of your XML document written down. There are two primary tools for doing this: XML Document Type Definitions (DTDs) and XML Schemas.

DTDs are the original tools for defining an XML-based language and are based on the way SGML languages are defined. Schemas are a newer development and allow for types of values to be defined in a broader fashion than DTDs allow. For a simple example such as this one, a DTD will suffice.

A DTD's purpose is to define exactly what types of elements and attributes can be used in a document and in which combination and structure they may be arranged. A DTD file looks somewhat similar to an XML or HTML file, but technically speaking, it's not XML because it doesn't follow the rules for XML; schemas, on the other hand, do follow the XML rules because the XML Schema Language is also an XML-based language.

An example of an XML DTD for the simple accessibility tip language is shown in Listing web2.2. You probably won't be able to understand everything unless you've worked with XML DTDs before, but the effect of this file is to determine what is allowable within the context of this XML-based language.

LISTING WEB2.2 A Simple DTD for This XML-Based Language

```
<!-- DTD for accessibility tip pages -->
<!ELEMENT tippage (accesstip)+>
<!ATTLIST tippage
    revision CDATA #REQUIRED
    xml:lang CDATA #REQUIRED
>
<!ELEMENT accesstip (headline, author, tipbody, tipexample*)>
<!ELEMENT headline (#PCDATA)*>
<!ELEMENT author (name, email?)>
<!ELEMENT name (#PCDATA)*>
<!ELEMENT email (#PCDATA)*>
<!ELEMENT tipbody (para+)>
<!ELEMENT para (#PCDATA ¦ code)*>
<!ATTLIST para
    paratype (normal¦note¦warning¦tip) #IMPLIED
>
<!ELEMENT code (#PCDATA)*>
<!ELEMENT tipexample (#PCDATA)*>
```

What does that mean? Here's some of what you can glean from the DTD about the structure of the document. This DTD defines a `<tippage>` element as consisting of one or more `<accesstip>` elements and requires that the `revision` and `xml:lang` attributes be set on `<tippage>`. Each `<accesstip>` contains a `<headline>`, an `<author>`, a `<tipbody>`, and zero or more `<tipexample>` elements. A `<tipbody>` holds one or more `<para>` tags, which themselves contain either normal text (`#PCDATA` in DTD terminology) or `<code>` elements. A `<para>` tag can optionally have a `paratype` attribute set, which can take one of four values.

Displaying XML

XML is quite useful for direct computer-to-computer communication. Using an agreed-upon common data format, a corporate website can communicate automatically with a partner company's site to exchange information. Instant messages can be marked up in an XML-based language for interoperability among messaging systems.

However, those aren't really of interest when we're talking about XML and CSS. More relevant to this book is the capability of Cascading Style Sheets to provide XML with the presentation layer that it lacks. HTML tags have built-in meaning and presentation styles, but XML tags don't, and that's where CSS styles come in handy.

Default Browser Display

If a browser understands the XML format, it displays an XML page as it displays an HTML page, except that it has no idea what the tags are, so the content alone is shown. Figure web2.1 shows how Safari displays the XML file from Listing web2.1.

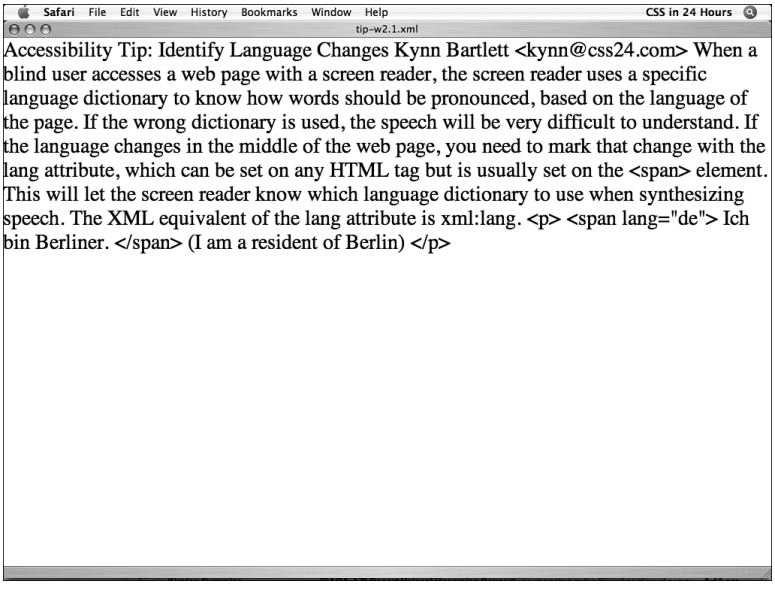

FIGURE WEB2.1
An XML file displayed by Safari.

Firefox does something more clever with XML files when displaying them. Recognizing that XML documents describe a hierarchical tree, Firefox shows unstyled XML files in a clickable tree structure. This is shown in Figure web2.2. You can click on a minus to close one branch of the tree or on a plus to open it up again.

All browsers choose one or another of these strategies when displaying XML files. Safari, Opera, and Lynx display the text content, whereas Internet Explorer—like Firefox—shows a tree structure.

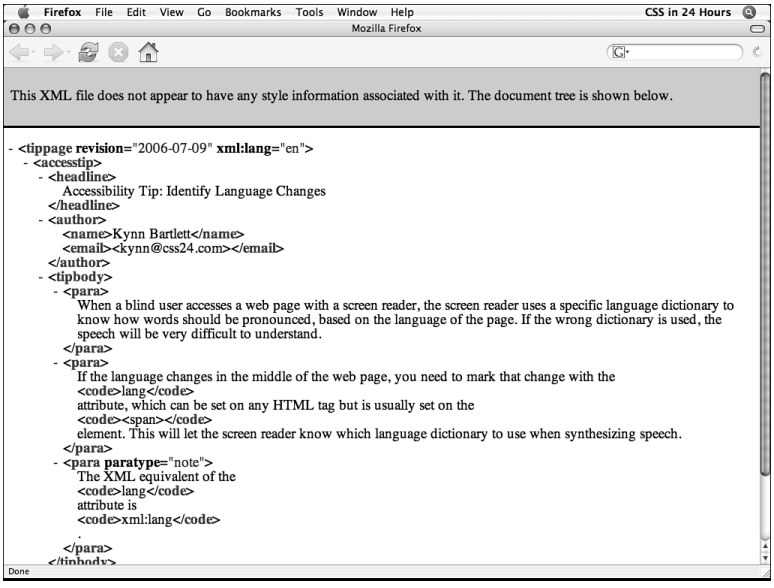

FIGURE WEB2.2
An XML file displayed by Firefox.

Linking Style Sheets in XML

Now, what you'd probably like to be able to do is to apply a style sheet to the XML file and use that to create a better presentation than the browser's default view. In HTML, you have three ways of associating CSS styles with content: linked style sheets (which use the `<link>` tag), embedded style sheets (which use the `<style>` element), and inline styles (which use the `style` attribute). They all depend on the fact that a tag or attribute has specific meaning in HTML.

XML doesn't provide any inherent meaning for any tags or attributes, so the HTML approach doesn't necessarily work for any generic XML document. Specific XML-based languages can be designed to have the equivalent of `<link>`, `<style>`, or `style`, but XML is meant to work with CSS even if the browser doesn't know what the specific tags and attributes represent.

The problem of linking CSS to XML is solved by using an XML *processing instruction* (PI for short). Processing instructions are, as the name implies, instructions to whatever program is processing the document and aren't actually part of the content itself. A processing instruction looks similar to an XML tag, but it has question marks directly inside the angle brackets. Processing instructions are not tags, which means that they don't ever have closing tags, although they have something similar to attributes to provide additional parameters.

The processing instruction for linking an external style sheet is called `xml-stylesheet`, and you write it like this:

```
<?xml-stylesheet type="text/css" href="filename"?>
```

As you can see, this parallels the `<link>` element of HTML in syntax and function. The `<?xml-stylesheet?>` processing instruction should be placed before your first element of the document, and you can have multiple style sheets if needed.

Styles for XML

CSS rules for XML elements are written just like the rules for HTML elements. The selector indicates to what part of the file the rule applies, and the declarations give values to properties.

Selectors for XML are the same as selectors for HTML; element names, attribute values, pseudo-classes, and relationship selectors can all be used in an XML rule. Property values, likewise, are the same as for HTML; you just have to remember that there are no default values already assigned to them. As an example, if you want a `<notice>` element to be styled as bold, red text in a block box, you simply write a rule like this:

```
notice { display: block;
         font-weight: bold;
         color: red; }
```

Although any CSS property and value can be used with XML, a number of properties are especially useful when designing style sheets for XML display, and later in this hour you learn how to use them most effectively.

A longer example of styles for XML is shown in Listing web2.3, which is a style sheet for displaying the accessibility tip XML document from Listing web2.1.

LISTING WEB2.3 A Style Sheet for Our Accessibility Tips

```
/* tip-w2.3.css */
tippage    { display: block;           font-size: medium;
             background-color: white;  color: navy;
             font-family: sans-serif; }
accesstip  { display: block;           margin: 1em;
             padding: 1em;             border: 2px solid black;
             background-color: #CCCCFF; }
headline   { display: block;           margin-bottom: 0.75em;
             font-size: x-large;       font-weight: bold;
             font-family: Verdana, sans-serif; }
author     { display: block;           margin-bottom: 0.75em;
             font-size: large;         font-weight: bold; }
name       { display: inline;          margin-right: 0.5em; }
email      { display: inline;          margin-right: 0.5em; }
tipbody    { display: block;           border: 2px solid white;
             padding: 0.5em;           margin-bottom: 0.75em; }
```

LISTING WEB2.3 Continued

```
para       { display: block;           margin-bottom: 0.65em;
             margin-top: 0.65em; }
para[paratype="note"]
           { border: 1px solid black;  padding: 1em; }
code       { display: inline;          font-family: monospace;
             color: black;             font-weight: bold; }
tipexample { display: block;           padding: 0.5em;
             border: 2px solid white;  margin-bottom: 0.75em;
             font-family: monospace;   white-space: pre; }
```

To use this style sheet with the XML file in Listing web2.1, you simply need to add the following line before the `<tippage>` tag:

```
<?xml-stylesheet type="text/css" href="tip-w2.3.css"?>
```

To see how the browser shows this file, look at Figure web2.3; it's quite different from the plain text look of Figure web2.1!

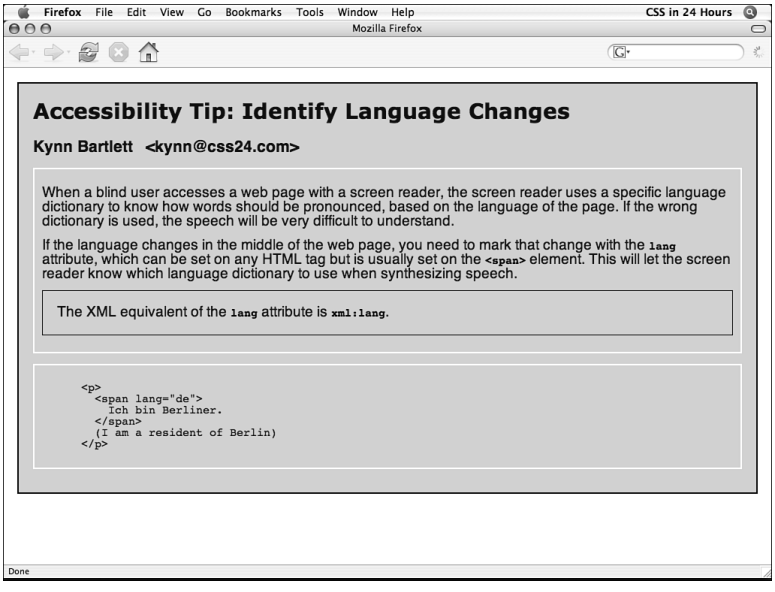

FIGURE WEB2.3 An XML file with a style sheet, displayed by Firefox.

Using `display` to Control Presentation

The `display` property is your biggest friend when using Cascading Style Sheets with XML because it's what you use to create `block` boxes. As a default, all elements are displayed as `inline` boxes, and they flow together into a mess, as shown in Figure web2.1. Using `display`, you can change these to the `block` value.

You can also use the `display` property to create lists, as covered in Hour 13, "Lists," by using the `display: list-item` value. This enables the list style properties to be applied to those elements.

Data tables can be displayed as HTML tables if you use the `display` values for tables, as discussed in Hour 17, "Styling Tables." This gives you the full range of columnar presentation supported by CSS in HTML.

Watch Out!

You should use table `display` values only for actual data tables, though; for layout, you should use positioning CSS, covered in Hour 19, "Absolute and Fixed Positioning," and Hour 20, "Page Layout in CSS."

Generating Content for XML Display

Because the raw content represented by XML files is often lacking in basic user interface clues, the capability to generate content is crucial when applying CSS directly to XML. The `:before` and `:after` pseudo-selectors and the `content` property—all introduced in Hour 23, "User Interface and Generated Content"—are extremely useful when working with XML.

Listing web2.4 is an additional style sheet to be added to the style rules shown in Listing web2.3 and applied to the accessibility tip XML document. The easiest way to do this is by simply adding a second processing instruction after the first, as follows:

```
<?xml-stylesheet type="text/css" href="tip-w2.4.css"?>
```

Alternately, an `@import` rule could be added to the beginning of the `tip-w2.3.css` style sheet.

LISTING WEB2.4 Additional Style Sheet with Generated Content

```
/* tip-w2.4.css */
author:before      { content: "Written by "; }
tipbody:before     { content: "Tip: ";
                     font-family: Verdana, sans-serif;
                     font-size: large; }
tipexample:before  { content: "Example: ";
                     font-family: Verdana, sans-serif;
                     font-size: large; }
para[paratype="note"]:before
                   { content: "Note: ";
                     font-weight: bold; }
```

These will add various bits of text content to the XML, so that the presentation makes a little more sense. Compare Figure web2.4 with Figure web2.3; it's much clearer, in respect to the generated content, what each section is meant to represent.

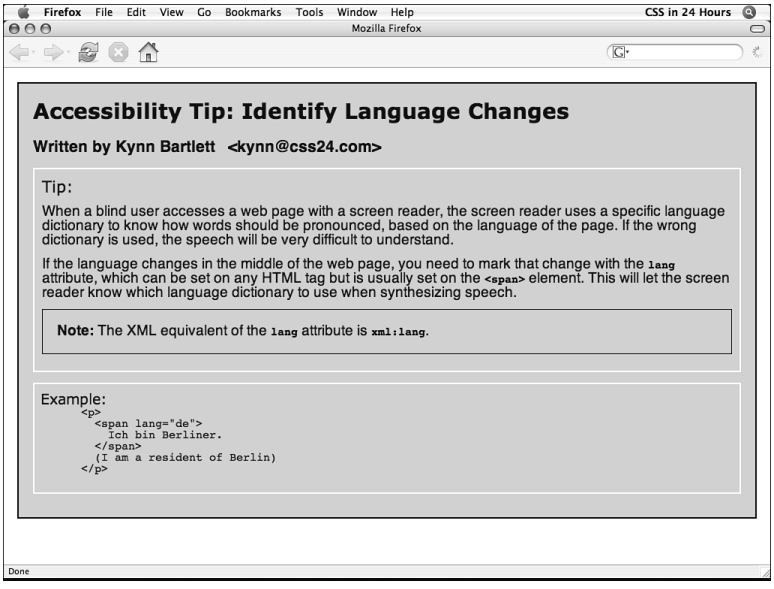

FIGURE WEB2.4
Firefox applies the updated style sheet to the XML.

XLink

As noted earlier, there's no intrinsic meaning to XML tags, which means there's no default presentation or behavior connected with them. In HTML, the <a> link means both "use the default presentation, usually blue underlined text" and "when this link is clicked, go to the address in the `href` attribute." In XML, CSS is used to provide the presentation, but the capability to define behaviors isn't part of the CSS language.

To address this need in XML, several additional specifications have been developed that create special tags and attributes, defining specific behavior or meaning in XML. To distinguish these from other tags or attributes you might create in your own language, the creators of these special XML tags and attributes—the World Wide Web Consortium—assigned them specific namespaces and namespace prefixes. A *namespace* is a unique URL that is associated with the specification, and a *namespace prefix* is associated with that URL and appended on the front of the tag or attribute.

The way to represent hypertext links and other types of document relationships in XML is to use XLink. The XLink specification defines several attributes related to the XLink namespace; these attributes are used to define relationships among data in XML.

You can use XLink to create a navigation bar for content, enabling you to link to related resources. XLink allows for simple and complex links; in this case, all you need are simple XLinks.

Watch Out!

Internet Explorer, Safari, and Opera do not support the simple XLink language, although Firefox does. This means that you are unable to create hypertext links in XML that function like the HTML <a> tag for users of those browsers.

Listing web2.5 is a revision of the previous XML file with a navigator bar added, complete with simple XLink attributes.

LISTING WEB2.5 An XML Document with XLinks

```
<?xml version="1.0"?>
<tippage xmlns:xlink="http://www.w3.org/1999/xlink"
         revision="2006-07-09" xml:lang="en">
  <accesstip>
    <!-- As before in Listing #web2#.1 -->
  </accesstip>
  <navbar>
    <navlink xlink:type="simple"
             xlink:href="http://kynn.com">
      Kynn's Home Page
    </navlink>
    <navlink xlink:type="simple"
             xlink:href="http://css24.com">
      CSS in 24 Hours
    </navlink>
    <navlink xlink:type="simple"
             xlink:href="http://www.w3.org/WAI/">
      Web Accessibility Initiative
    </navlink>
    <navlink xlink:type="simple"
             xlink:href="http://www.webaim.org">
      WebAIM
    </navlink>
  </navbar>
</tippage>
```

The effect of the `xlink:type` attribute is to declare the `<navlink>` elements to be part of a relationship link. In this case, they are a simple link that goes from the `<navlink>` to an external resource indicated by an `xlink:href` attribute. The end result is a link that is functionally the same as an `<a href>` link in HTML. Browsers that understand XLink should treat a `<navlink>` the same as an `<a>` link. Styles can be added to display this link in various ways, as well.

Styling XLink Links

When you style a normal HTML link, you use a selector on the <a> element that is modified by a pseudo-class selector, such as :link, :visited, :active, :hover, or :focus. When styling an XLink, the approach is much the same. A browser that understands XLink will set the appropriate pseudo-class states on XLinks. This makes it possible to write style rules with selectors such as navlink:link or navlink:active.

Listing web2.6 is a style sheet that is designed to be applied to the extended version of the accessibility tip XML, along with the tip-w2.3.css and tip-w2.4.css style sheets. To use these, three processing instruction lines are added to the XML document shown in Listing web2.5, which is the longer tip file with the navigation bar. Those lines are

```
<?xml-stylesheet type="text/css" href="tip-w2.3.css"?>
<?xml-stylesheet type="text/css" href="tip-w2.4.css"?>
<?xml-stylesheet type="text/css" href="tip-w2.6.css"?>
```

LISTING WEB2.6 Style Sheet for XLink Navigation Bar

```
/* tip-w2.6.css */
accesstip       { position: absolute;     left: 200px;
                  top: 0px; }
navbar          { display: block;          position: absolute;
                  left: 0px;               top: 0px;
                  width: 150px;            margin: 1em;
                  border: 2px solid black; padding: 0.5em;
                  background-color: #FFFFCC; }
navbar:before   { font-size: large;        content: "Links: ";
                  font-weight: bold;
                  font-family: Verdana, sans-serif; }
navlink         { display: block;          font-size: small;
                  font-weight: bold;       text-align: center;
                  margin: 0em 0.4em;
                  font-family: Verdana, sans-serif; }
navlink:link    { color: blue; }
navlink:visited { color: purple; }
navlink:hover   { color: red; }
navlink:active  { color: red; }
```

You create the navigation bar by using absolute positioning to place both the <accesstip> element and the <navbar> element in their appropriate locations. Pseudo-classes are used to set link effects, and a little extra content is generated at the start of the navigation bar. The full effect is shown in Figure web2.5.

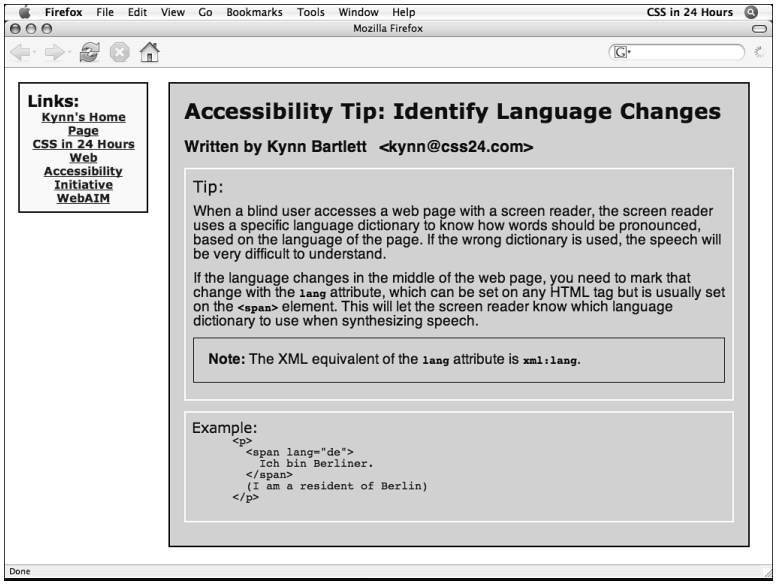

FIGURE WEB2.5
Firefox displays the XLinks and CSS.

XML-based Languages and CSS

The first part of this hour described how you can apply Cascading Style Sheets to generic XML pages—those that are not necessarily part of a language known by the browser but which nevertheless conform to the rules of XML. The `<?xml-stylesheet?>` processing instruction is a universal method for applying CSS to any XML file.

However, if you are dealing with an XML-based language where the semantics are known—meaning that the authors of the document and the browser (or other software) both understand what the tags mean—the rules could be very different, depending on how the language has decided to use CSS.

In this section, you'll survey some of the XML-based languages that use Cascading Style Sheets and see what role CSS plays in those languages.

XHTML

Extensible Hypertext Markup Language 1.0 (XHTML) is simply HTML 4.01 written in accordance with the rules for XML. All tags are the same case (lowercase), all attributes are quoted, and all tags are explicitly closed. Like HTML 4.01, XHTML 1.0 comes in three flavors—Strict, Transitional, and Frameset.

Did you Know?

If you want to convert from HTML to XHTML, a good utility is the HTML Tidy program available from the W3C. This can perform a number of functions, from cleaning up your HTML code to changing to the proper XHTML syntax. You can download the program for free from http://www.w3.org/People/Raggett/tidy/.

The primary advantage of XHTML is that it's both XML and HTML at the same time, meaning that you can use it with XML applications for greater interoperability, and you can also use it with existing HTML browsers.

In addition, further development on "non-X" HTML by the World Wide Web Consortium has been stopped; all future work on HTML will be done as XHTML. One example of this work is the *modularization* of XHTML, which divides XHTML tags and attributes into sets called *modules*. Each module has a specific function, and groups of modules can be combined together to build new XHTML languages. XHTML 1.1 is the newest version of HTML, built from XHTML modules.

XHTML version 1.1 is based on Strict HTML, which means that there are no presentation attributes or elements; instead, XHTML 1.1 relies entirely on CSS for presentation effects. Future versions, including XHTML 2.0, will continue this trend, making knowledge of CSS essential for future XHTML development.

In all versions of XHTML, you can apply CSS as you do in HTML; the `<link>` and `<style>` tags and the `style` attribute are defined as in HTML.

SVG

The Scalable Vector Graphics (SVG) language is an XML-based format for creating vector graphics. Vector graphics, unlike bitmapped graphical formats such as GIF or JPEG, can be scaled up and down in size without loss of resolution, and they are often much smaller than an equivalent bitmap. SVG is a W3C specification developed by the graphics working group.

SVG files use Cascading Style Sheets properties to define color, text effects, fonts, and other presentation qualities. Like HTML, SVG has predefined semantics for linking and embedding style sheets, as well as for inline styles.

XUL

XML-based User Interface Language (XUL) is a language developed by the Mozilla project for use with the Mozilla and Netscape 6 Web browsers. Rather than being a content language, XUL describes the user interface of a program, including the appearance and colors, the menus, and the buttons. Using XUL, browser users can create skins that customize the function and look of their user interfaces. XUL uses

Cascading Style Sheets extensively to provide formatting effects on user interface components and is an example of CSS used for something besides simply styling of web content.

XSL

Extensible Style Sheet Language (XSL) is a broad term that actually covers two related technologies, XSL Formatting Objects (XSL-FO) and XSL Transformations (XSLT).

The XSL-FO language describes the end appearance of a document in an XML-based syntax. This is quite useful for a fixed layout, such as on the printed page, but is not as useful on the screen. Formatting objects are XML elements describing specific areas of the printed page and the content contained by them; the formatting objects dictate the ultimate appearance of the document. XSL-FO elements and attributes are based on Cascading Style Sheets properties, and so the transition from CSS to XSL-FO is not particularly hard after you learn the XML-based syntax.

The XSLT language was written to transform an arbitrary XML document, such as the accessibility tip, into XSL-FO. XSLT has evolved beyond this single purpose, however, and can be used for any kind of transformation where you want to convert from one XML-based language to another. For example, you could write an XSL Transformation to change the accessibility tip XML file into an XHTML page, with CSS rules for the presentation effects. Although XSLT does not use style sheets directly, you can easily use CSS and XSLT together to produce custom presentation effects.

Summary

Extensible Markup Language (XML) is not a replacement for HTML; instead, it is a meta-language for creating new languages that can be used on the Web and in computer applications. XML is defined by a strict set of rules, including "tags must nest properly" and "each tag must have a matching closing tag." A document that conforms to these rules is considered proper XML.

XML languages are markup languages that conform to the rules of XML. A language can be formally defined by using an XML DTD or Schema to specify which tags and attributes can be used, but such formal definition is optional. The tags of an XML language don't necessarily have any inherent presentational styles associated with them, and a browser often displays an XML file as plain text content or as a structured tree.

CSS can be used with XML just as it is used with HTML; the properties are the same, as is the way selectors are used. Applying CSS to XML builds a presentation style

that can make the structure of the XML content easier to understand. CSS properties for positioning, display, and generated content are especially useful with XML.

XML-based languages, such as Extensible Hypertext Markup Language (XHTML), Scalable Vector Graphics (SVG), XML-based User Interface Language (XUL), and Extensible Style Sheet Language (XSL), were created to work with Cascading Style Sheets. As the technology of web design continues to evolve along the path of XML, the CSS knowledge you've gained from this book will continue to serve you well!

Workshop

The workshop contains a Q&A section, quiz questions, and exercises to help reinforce what you've learned in this hour. If you get stuck, the answers to the quiz can be found after the questions.

Q&A

Q. ***I read the CSS level 2 specification and it says to use*** `<?XML:STYLESHEET?>` ***not*** `<?xml-stylesheet?>`***. What's up with that?***

A. The CSS level 2 specification was written before the method of associating style sheets with XML documents was formalized. That part of the CSS2 recommendation has been superceded by later specifications that specifically address the relationship between XML and CSS.

Q. ***Do current web browsers support XHTML?***

A. Yes and no. Few of them were coded specifically for XHTML, but if you write your XHTML in accordance with backward-compatibility rules, HTML browsers can understand it. The XHTML backward-compatibility rules are listed in the HTML 1.0 specification at http://www.w3.org/TR/xhtml1.

Q. ***You said that all XHTML tags are lowercase, but I write my HTML tags as uppercase. Why do I have to write XHTML in lowercase?***

A. Because XML is case sensitive—unlike HTML. XHTML tags have to match and must be written consistently with respect to the case of the characters. When XHTML was created, the decision was made to use lowercase letters instead of uppercase. Why? It nearly came down to a coin-toss, and effectively it was an arbitrary decision. Either way, half of the people would be disappointed! So lowercase letters were chosen, and that's what we use for writing XHTML. If you use lowercase letters already, you're in luck. If not, you'll just have to get used to it…

Q. ***What does XSLT let me do that CSS does not?***

A. Using XSLT, you can affect the structure of the document, not just the appearance, as you can with CSS. For example, you can write an XSLT transformation that extracts specific content and repurposes it in a completely new manner, such as creating a summary of the hypertext links on a page. If you then apply CSS to the resulting XML (or XHTML), you can make dynamic custom interfaces.

Quiz

1. Each of the following is invalid according to the rules of XML. What rule does each one violate?

   a. `<author><name>Kynn Bartlett</author></name>`

   b. `<catalog code=r343>CSS in 24 Hours</catalog>`

   c. `<animation src="glow.mov">`

   d. `<timestamp>24-Jun-2002</TimeStamp>`

2. Consider the following very simple XML file:

```
<?xml version="1.0">
<friends>
  <person>
    <name>Russ Smith</name>
    <age>33</age>
  </person>
  <person>
    <name>Nick Mamatas</name>
    <age>30</age>
  </person>
</friends>
```

   How would you write a style sheet that displays this as a simple table?

3. Which XML-based language uses CSS to define the colors and text properties of graphics?

   a. XUL

   b. SVG

   c. XHTML

   d. XSL-FO

Answers

1. These all break at least one rule of XML. In (a), the tags aren't nested properly. In (b), the attribute value is not enclosed in quotes. There is no closing tag in (c), and the closing tag doesn't match the case of the opening tag in (d).

2. You'll need to use the table-related `display` values to do this effectively. Here is a simple style sheet that does that:

```
friends   { display: table;
            border: 3px inset black; }
person    { display: table-row; }
name, age { display: table-cell;
            padding: 0.5em;
            border: 2px inset gray; }
age:after { content: " years old"; }
```

3. The correct answer is (b), Scalable Vector Graphics.

Exercises

A number of new technologies were introduced in this hour that are beyond the scope of this book to cover in sufficient detail. If you're interested in learning more, here are some resources to get you started:

- Learn more about XML by visiting the W3C's XML pages at http://www.w3.org/XML/ or by reading *Sams Teach Yourself XML in 24 Hours, Complete Starter Kit, 3rd Edition;* ISBN: 067232797X.
- XHTML is covered in *Sams Teach Yourself HTML and XHTML in 24 Hours, 6th Edition*; ISBN: 0672325209, or you can learn more from the W3C's XHTML page at http://www.w3.org/Markup/. Be sure to check out the free HTML Tidy program!
- A good site for Scalable Vector Graphics is http://www.w3.org/Graphics/SVG/, and you can also read about them in *Sams Teach Yourself SVG in 24 Hours*; ISBN: 0672322900.
- The definitive source for XUL information is the Mozilla website at http://www.mozilla.org/.
- Information on XSL, XSL-FO, and XSLT can be found at http://www.xslinfo.com/, as well as on the W3C's site.