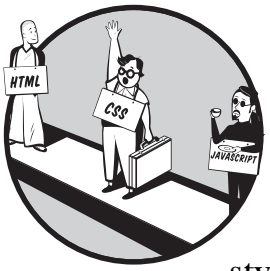
1

CASCADING STYLE SHEETS



Once you add HTML elements to a page, positioning and formatting take center stage. At first, CSS consisted of two parts: CSS1 and CSSP. CSS1 was used for defining style rules in a style sheet, and CSSP was used to position elements on a page. However, the W3C consortium later decided to combine both parts into one, known as CSS2, and this book follows the recommendations of the CSS2 standard. CSS2 provides control over two important functions:

The style used to format the contents of a tag With CSS, the formatting of a tag can be defined independently of its content. For example, the `<table>` tag tells the browser that a table has to be placed on the page. However, the formatting and style of the `<table>` tag can be defined separately in a style sheet. This makes CSS scripting simple and concise.

The positioning and dimensions of elements on a page CSS will also allow you to set the size of an element, such as a `<table>` tag, as well as its precise location on a web page.

Style Sheets

Style sheets contain style rules that govern the formatting and positioning of specific elements on a page. Style sheets can be placed in different locations:

External The style sheet is located in a text document separate from the web pagedocument.

Internal The style sheet is defined in the `<head>` section of the page source code. (This is also known as an *embedded* style sheet.)

Inline Using the `style` attribute, the style sheet is defined inside the `<html>` tag that is being formatted.

External Style Sheets

Style sheets can reside on the server as a text or JavaScript document linked to from the web page document. For example, a style sheet document called myStyleSheet.css, stored in the same directory as the page, would be referenced in the `<head>` section of the page as follows:

```
<head>
<link rel="stylesheet" type="text/css" href="myStyleSheet.css">
</head>
```

Do not confuse the `<link>` element with the `<a>` element; the `<link>` element must be specified in the `<head>` section of the page and requires the `rel`, `type`, and `href` attributes. `rel` identifies the type of document, “stylesheet” in the preceding example; `type` identifies the type of content in the linked document, “text/css” in the example; and `href` identifies the location of the document.

This method can be used to link large style sheets, or when several HTML documents will be using the same style sheet. If the size of the style sheet is small and is not used by multiple HTML documents, the style sheet rules can be included inside the `<head>` section of the page, as is explained in the next section.

Using the @import Rule to Connect with External Style Sheets

The `@import` at-rule allows the page to import a CSS document inside the `<style>` element tag without requiring the use of the `<link>` element in the `<head>` section of the page. The `@import` at-rule uses the URL of the document in place of the `<link>` element’s `href` attribute. Here’s an example:

```
<style type="text/css">
@import url(myStyleSheet.css)
</style>
```

As you can see in this example, you can use the `@import` functionality in place of `<link>` when referencing an external style sheet. However, there are no distinct advantages in using one or the other.

NOTE *All rules that start with the @ symbol are known as "at-rules." Another useful at-rule is* `@font-face`*, which allows you to download and use fonts in the page that the browser does not already have installed. Only TrueType fonts in an .eot type file format can be downloaded using this method.*

Internal Style Sheets

If a style sheet contains only a few lines of code, then it can be inserted directly in the `<head>` section of the page using the `<style>` element tag. Although the `<style>` element tag can be placed in the `<body>` section of the page as well, it is a better practice to include it in the `<head>` section. For example,

```
<head>
<style type="text/css">
p {color:red; font-size:3; font-family:Verdana}
h1 {color:blue}
/* more style rules can go here */
</style>
</head>
```

The preceding example contains two rules — one applicable to the `<p>` element, and one applicable to the `<h1>` element. Style sheet rules are preceded by the type of element to which they are applied (`<p>`, `<h1>`, and so on), and they are enclosed in brackets {}.

Don't confuse the `style` attribute with the `<style>` element. While the `<style>` element can contain several rules applied to several tags and must have a closing `<style>` tag, the `style` attribute is defined *inside* an element's tag using an inline style sheet.

Inline Style Sheets

You can define a single style rule inside an element's tag by setting the element's `style` attribute. For example,

```
<p style="font-family:Verdana; font-size:3; color:red"> . . . </p>
```

This example has a single rule that sets three style attributes for the `<p>` element being defined: the font is set to `Verdana`, the font size is set to `3`, and the font color is set to `red`.

Style Sheet Summary

You can use style sheets in an HTML document in four ways:

- Use the <link> element inside the <head> section.
- Use the @import at-rule in the <style> element tag defined inside the <head> section of the document.
- Insert a <style> element tag inside the <head> section of the document.
- Insert the style attribute within an HTML tag.

Declarations

The combination of an attribute like text-size with its corresponding value, say 14pt, is known as a *declaration*. When defining a style rule, you must separate multiple declarations with semicolons. For example,

```
{ text-color:red; font-size:12pt }
```

This example contains two declarations; one sets the text color to red, and the other sets the font size to 12pt.

Selectors

Selectors allow you to apply rules to elements wherever they are used on the page. You can use selectors only when using external or internal style sheets.

There are simple selectors, like h1, and there are contextual selectors, which consist of several simple selectors, like h1 h2. The syntax is as follows:

```
element1 {style rule} // simple selector
element1 element2 {style rule} // contextual selector
```

Here is an example using the <h1> and <p> elements:

```
h1 {style rule}
h1 p {style rule}
```

Simple Selectors

A *simple selector* defines a style rule for a single element type (such as <h1>, <b>, or <p>), and the rule is applied to all elements of that particular type. For example,

```
p { color:red }
```

This rule specifies that the color of the text inside *all* <p> elements be red.

Contextual Selectors

Contextual selectors allow you to define the same style rule for multiple element types. Combining elements containing the same rule avoids repeating the same code for different elements. For example,

```
h1 em ul { color:red }
```

This rule specifies that the color of all text inside <h1>, , and <ul> tags should be red. Each simple selector is separated by a blank space, thus forming the contextual selector. The preceding example is equivalent to the following three lines of code:

```
h1 { color:red }
em { color:red }
ul { color:red }
```

Using the class Attribute as a Selector

There is yet another way to apply a selector to various elements that do not share the same tag. You can define a rule with a class selector that can then be used by the class attribute available to all elements. Each separate class you define must have a unique name.

In the style sheet, the name of the class needs to have a period in front of it so that the browser recognizes it as a class. For example,

```
.wide { color: red }
```

In this example, all the elements that have a class attribute with a value of wide will contain the color red.

In the following example, the <p> element tag references the wide class:

```
<p class="wide">contents</p>  // the word "contents" will be red.
```

It is also possible to define different styles for different element types that have class attributes sharing the same value. For example, to give <p> elements of a particular class the color of blue, while giving all other elements of the same class a color of red, you could use the following code:

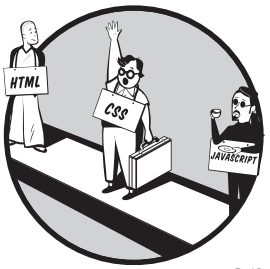
```
.colored {color: red}
p.colored {color: blue}
```

In the preceding example, all <p> (paragraph) tags that have the class attribute of colored will turn blue, and all other elements using this class will turn red. All <p> elements that do not contain a class attribute with a value of colored will not be affected by either class declaration.

NOTE *Only one class can be specified per selector.*

11

JAVASCRIPT PROPERTIES



This chapter provides a comprehensive list of JavaScript properties. Each property contains compatibility information, a brief description, its syntax, an example, and an applies-to list of HTML elements, JavaScript collections, and JavaScript objects that make use of this property.

In order to distinguish between the HTML elements, JavaScript collections, and JavaScript objects that appear in the applies-to lists, the following conventions have been used:

- Angle bracket tags (< and >) surround the names of HTML elements (for example, `<html>`).
- An asterisk follows the names of JavaScript collections (`areas*`).
- The names of JavaScript objects appear without adornment (`window`).

NOTE *See Chapter 4 for an introduction to JavaScript properties.*

abbr

Compatibility: NN6, NN7, IE6

Read and write property. Specifies an abbreviated text for the element. It can be used by alternate means of rendering content, such as speech synthesis or Braille.

Syntax:

```
document.getElementById("elementID").abbr = value
document.all.elementID.abbr = value // IE only
```

Example:

```
<html><head><script language="JavaScript">
function goAbbr() {document.all.myTH.abbr = "Abbreviation";}
</script></head>
<body onLoad="goAbbr();">
<table width="428" border="1" cellspacing="5" cellpadding="5"
bordercolor="#0000FF">
    <th id="myTH" colspan="2">This is the table heading </th>
    <tr><td> Cell 1 content </td><td> Cell 2 content </td></tr>
    <tr><td> Cell 3 content </td><td> Cell 4 content </td></tr>
</table>
</body></html>
```

Applies to:

```
<td>, <th>
```

accept

Compatibility: NN6, NN7, IE6

Read and write property. Specifies the comma-separated list of MIME content types that the element can accept. Values: `text/html`, `image/png`, `image/gif`, `video/mpeg`, `audio/basic`, `text/tcl`, `text/javascript`, and `text/vbscript`.

Syntax:

```
document.getElementById("inputID").accept = value
document.all.inputID.accept = value // IE only
```

Example:

```
<html><head><script language="JavaScript">
function goAccept() { document.all.myB.accept = "image/gif"; }
</script></head>
<body bgcolor="#FFFFFF" text="#000000" onLoad="goAccept();">
<input type="text" name="textfield" size="50" accept="image/gif"
```

```
value='The content type of this field is "text/html"'>
<input type="button" id="myB" value='The content type for this button is "image/
gif"'>
</body></html>
```

Applies to:

```
<input>
```

acceptCharset

Compatibility: NN6, NN7, IE5, IE5.5, IE6

Read and write property. Specifies a comma- or space-separated list of character sets that the server receiving form input must support. The UTF-8 character set will be used if the server does not support the character set sent by the document.

Syntax:

```
document.getElementById("formID").acceptCharset = value
document.all.formID.acceptCharset = value // IE only
```

Example:

```
<html><head><script language="JavaScript">
function goAcceptCharset() {alert(document.all.myForm.acceptCharset);}
</script></head>
<body onLoad="goAcceptCharset();">
<form id="myForm" method="post" action="" acceptcharset="UTF-8">
<!--input elements go here-->
</form>
</body></html>
```

Applies to:

```
<form>
```

accessKey

Compatibility: NN6, NN7, IE4, IE5, IE5.5, IE6

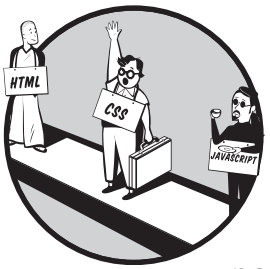
Read and write property. Specifies the one-character accelerator key for the element. Pressing the ALT key and the accelerator key simultaneously brings the element into focus.

Syntax:

```
document.getElementById("elementID").accessKey = value
document.all.elementID.accessKey = value // IE only
```

12

JAVASCRIPT METHODS



JavaScript methods are the verbs of the JavaScript language. Each method in this chapter contains compatibility information, a brief description, syntax information, parameter values (where applicable), a functional example, and an applies-to list identifying all the HTML elements, JavaScript collections, and JavaScript objects that the method can be used with.

In order to distinguish the HTML elements, JavaScript collections, and JavaScript objects that appear in the applies-to lists, the following conventions have been used:

- Angle bracket tags (< and >) surround the names of HTML elements (for example, `<html>`).
- An asterisk follows the names of JavaScript collections (`areas*`).
- The names of JavaScript objects appear without adornment (`window`).

NOTE *See Chapter 4 for an introduction to JavaScript methods.*

abs()

Compatibility: NN4, NN6, NN7, IE4, IE5, IE5.5, IE6

This method returns the absolute value for the number specified.

Syntax:

```
Math.abs(param1)
```

param1 Required; the number to convert.

Example:

```
<button onclick="alert(Math.abs(-10));">ABS</button>
```

Applies to:

```
Math
```

acos()

Compatibility: NN4, NN6, NN7, IE4, IE5, IE5.5, IE6

This method returns the arccosine value for the number specified.

Syntax:

```
Math.acos(param1)
```

param1 Required; the number to convert.

Example:

```
<button onclick="alert(Math.acos(1));">ACOS</button>
```

Applies to:

```
Math
```

add() (1)

Compatibility: IE4, IE5, IE5.5, IE6

This method adds an element to the collection.

Syntax:

```
document.all.selectID.add(param1, param2)
collectionName.add(param1, param2)
```

param1 Required; the element to add.

param2 Optional; the index position for the added element.

Example:

```
<script language="JavaScript">
function B1() {
var newOption = document.createElement('<option value="TOYOTA">');
document.all.mySelect.options.add(newOption);
newOption.innerText = "Toyota";}
function B2() {document.all.mySelect.options.remove(0);}
</script>
<select id="mySelect">
<option value="HONDA">Honda</option>
<option value="ACURA">Acura</option>
<option value="LEXUS">Lexus</option>
</select>
<input type="button" value="Add" onclick="B1();">
<input type="button" value="Remove" onclick="B2();">
```

Applies to:

```
areas*, controlRange*, options*, <select>
```

add() (2)

Compatibility: IE5.5, IE6

This method adds a new `namespace` object to the `namespaces` collection.

Syntax:

```
namespaces.add(param1, param2, param3)
```

param1 Required; the name of the namespace to add.

param2 Required; the URN of the namespace.

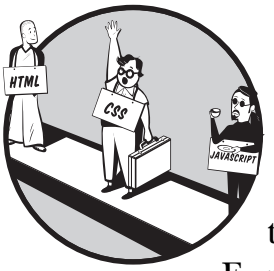
param3 Optional; the URL of the behavior to be added to the namespace.

Example:

```
<html xmlns:firstNS>
<head>
<?import namespace="firstNS" implementation="someFile1.htc">
<script>
function addNS(){
namespaces.add(secondNS, someFile2.htc);
namespaces.add(thirdNS, someFile3.htc);
}
</script></head>
```

15

HTML+TIME MICROSOFT TECHNOLOGY



The HTML+TIME (Timed Interactive Multimedia Extensions) Microsoft technology was first introduced in Internet Explorer 5.0, and it is not compatible with any browsers other than Internet Explorer. When other browsers encounter an HTML element with an attached HTML+TIME behavior, they ignore the behavior and render the element normally.

HTML+TIME is essentially a Microsoft behavior. In Internet Explorer versions 5.0 and later, HTML+TIME adds timing and media synchronization support to HTML pages, meaning that you can now use HTML to quickly and easily create multimedia-rich interactive presentations with little or no scripting. There are two versions of this technology:

HTML+TIME 1.0 The first version was only made available through XML documents, while HTML and DHTML document implementation was not supported. The first version was compatible with Internet Explorer 5.0, and it was accessed using the `time` behavior.

HTML+TIME 2.0 This version was made available not only to XML documents, but to HTML and DHTML documents as well. Version 2.0 is compatible with Internet Explorer 5.5 and later, and it is accessed using the `time2` behavior. The syntax throughout this chapter is based on the `time2` behavior.

With special HTML+TIME HTML elements and attributes, you can specify when an element appears on a page, how long it remains displayed, and how the surrounding HTML elements are affected. For example, you can specify if and when a sound file should start playing, when it should stop, and how many times it should repeat.

Using this Microsoft technology, HTML elements can also be grouped together, allowing a single timing effect to act on all the elements in the group, either simultaneously or sequentially.

The following steps will allow you to add an HTML+TIME behavior to an HTML page:

1. Create the XML namespace `:t` by declaring it inside the HTML tag, as follows:

```
<html xmlns:t="urn:schemas-microsoft-com:time">
```

2. Introduce the HTML+TIME behavior of interest in one of two ways. Either introduce it as an inline style attribute of the element that is to be affected by the behavior, as follows:

```
<element style="behavior:url(#default#time2)">
```

   Or introduce the behavior in a style sheet and then reference the class in the `class` attribute of the element that is to be affected by the behavior, as follows:

```
<style>.time{behavior:url(#default#time2);}</style>
<element class="time">
```

3. Establish `t:` as the namespace, and import the `time2` behavior into the namespace as follows:

```
<?import namespace="t" implementation="#default#time2">
```

4. Specify the beginning and ending times for the behavior. This is accomplished by specifying the `begin` and `end` attributes in the affected element.
5. (Optional.) Specify an action to take while the element is active on the timeline by adding the desired attribute.

6. (Optional.) If you want to be able to access the created HTML+TIME element in JavaScript, you must expose the element by setting its `id` attribute using the following syntax:

```
<t:element id="stringID"/>
```

Organization of This Chapter

As stated in the introduction to Part II of this book, this chapter contains all of the information related to the HTML+TIME technology. Rather than distributing all of the HTML+TIME-related HTML elements, HTML attributes, events, JavaScript properties, JavaScript methods, JavaScript collections, and JavaScript objects throughout Chapters 5, 6, 7, 11, 12, 13, and 14, respectively, all of this information is organized here in one place.

However, in the interest of consistency, the sections in this chapter are organized in the same order as the reference chapters listed above. Therefore, you will encounter HTML+TIME-compatible HTML elements first, followed by the compatible HTML attributes (and JavaScript properties, because they are related), events, JavaScript methods, JavaScript collections, and JavaScript objects.

HTML Elements

All of the HTML+TIME HTML elements are listed here in the same manner that the non-HTML+TIME HTML elements are listed in Chapter 5. For each element, you will find a compatibility listing, a description, a syntax listing, and an example, followed by lists of the compatible HTML+TIME HTML attributes, events, JavaScript properties, JavaScript methods, JavaScript collections, and JavaScript objects.

<t:animate/>

Compatibility: IE5.5, IE6

This element animates the attribute specified by `attributeName` of the element specified by `targetElement`. The animated attribute must have an initial value when the `<t:animate/>` element is loaded.

Syntax:

```
<t:animate attributes events/>
```

Example:

```
<html xmlns:t="urn:schemas-microsoft-com:time">
<head><?import namespace="t" implementation="#default#time2"></head>
<body><div id="myDiv" style="position:absolute; top:50px; left:100px; width:200px;
height:100px; background-color:blue"></div>
```