Start Code	End Code	Block Name
\u2D30	\u2D7F	Tifinagh
\u2D80	\u2DDF	Ethiopic Extended
\u2E00	\u2E7F	Supplemental Punctuation
\u2E80	\u2EFF	CJK Radicals Supplement
\u2F00	\u2FDF	KangXi Radicals
\u2FF0	\u2FFF	Ideographic Description Characters
\u3000	\u303F	CJK Symbols and Punctuation
\u3040	\u309F	Hiragana
\u30A0	\u30FF	Katakana
\u3100	\u312F	Bopomofo
\u3130	\u318F	Hangul Compatibility Jamo
\u3190	\u319F	Kanbun
\u31A0	\u31BF	Extended Bopomofo
\u31C0	\u31EF	CJK Strokes
\u31F0	\u31FF	Katakana Phonetic Extensions
\u3200	\u32FF	Enclosed CJK Letters and Months
\u3300	\u33FF	CJK Compatibility
\u3400	\u4DBF	CJK Unified Ideographs Extension A
\u4DC0	\u4DFF	Yijing Hexagram Symbols
\u4E00	\u9FBB	CJK Unified Ideographs
\uA000	\uA48F	Yi Syllables
\uA490	\uA4CF	Yi Radicals
\uA700	\uA71F	Modifier Tone Letters
\uA720	\uA7FF	Latin Extended-D
\uA800	\uA82F	Syloti Nagri
\uA840	\uA87F	Phags-pa
\uAC00	\uD7A3	Hangul Syllables
\uD800	\uDB7F	High Surrogates
\uDB80	\uDBFF	Private Use High Surrogates
\uDC00	\uDFFF	Low Surrogates

continued

TABLE 15-1 *(continued)*

Start Code	End Code	Block Name
\uE000	\uF8FF	Private Use Area
\uF900	\uFAFF	CJK Compatibility Ideographs
\uFB00	\uFB4F	Alphabetic Presentation Forms
\uFB50	\uFDFF	Arabic Presentation Forms-A
\uFE00	\uFE0F	Variation Selectors
\uFE10	\uFE1F	Vertical Forms
\uFE20	\uFE2F	Combining Half Marks
\uFE30	\uFE4F	CJK Compatibility Forms
\uFE50	\uFE6F	Small Form Variants
\uFE70	\uFEFF	Arabic Presentation Forms-B
\uFF00	\uFFEF	Halfwidth and Fullwidth Forms
\uFFF0	\uFFFF	Specials

ISO-8859-1

If you are working on websites for Western audiences, you will most likely use ISO-8859-1, which, although not officially a subset of UTF-8, does map to the Latin Basic and Latin Extended A Unicode sets.

The most familiar encoding to Western HTML developers, ISO-8859-1, is a subset of Unicode and can be used safely because most modern user agents support Unicode. Although ISO-8859-1 is not part of the Unicode standard, the two bodies governing both standards have worked together to standardize the models.

Note

The entire set of ISO-8859-1 numeric references can be found at `www.w3 .org/MarkUp/html3/latin1.html`. ■

Table 15-2 shows the entities you are likely to encounter as an HTML developer. If your encoding is UTF-8, then you can use the decimal references, but for compatibility with older user agents you should use HTML entities, because many older user agents don't support Unicode.

Latin-1 Supplement (U + 00 C0 - U + 00FF)

The Latin-1 Supplement also contains values from ISO-8859-1. The characters in this Unicode block are used for the following languages:

- Danish
- Dutch

- Faroese
- Finnish
- Flemish
- German
- Icelandic
- Irish
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

It extends the Basic Latin encoding with a miscellaneous set of punctuation and mathematical signs.

TABLE 15-2

ISO-8859-1 HTML Entities

Description	Decimal-Based Code Value	HTML Entity	How Character Appears on Web Page
Quotation mark	&#34;	&quot;	"
Ampersand	&#38;	&amp;	&
Less-than sign	&#60;	&lt;	<
Greater-than sign	&#62;	&gt;	>
Nonbreaking space			
Inverted exclamation	¡	&iexcl;	¡
Cent sign	¢	&cent;	¢
Pound sterling	&#163;	&pound;	£
General currency sign	&#164;	&curren;	¤
Yen sign	&#165;	&yen;	¥
Broken vertical bar	&#166;	&brvbar;	¦
Section sign	&#167;	&sect;	§
Umlaut (diaeresis)	&#168;	¨ &dia;	¨
Copyright	&#169;	&copy;	©

continued

TABLE 15-2 *(continued)*

Description	Decimal-Based Code Value	HTML Entity	How Character Appears on Web Page
Feminine ordinal	&#170;	&ordf;	ª
Left angle quote, guillemotleft	&#171;	&laquo;	«
Not sign	&#172;	&not;	¬
Soft hyphen	&#173;	&shy;	-
Registered trademark	&#174;	&reg;	®
Macron accent	&#175;	&macr;	¯
Degree sign	&#176;	&deg;	°
Plus or minus	&#177;	&plusmn;	±
Superscript two	&#178;	&sup2;	²
Superscript three	&#179;	&sup3;	³
Acute accent	&#180;	&acute;	´
Micro sign	&#181;	&micro;	µ
Paragraph sign	&#182;	&para;	¶
Middle dot	&#183;	&middot;	·
Cedilla	&#184;	&cedil;	¸
Superscript one	&#185;	&sup1;	¹
Masculine ordinal	&#186;	&deg;	º
Right angle quote, guillemotright	&#187;	&raquo;	»
Fraction one-fourth	&#188;	&frac14;	¼
Fraction one-half	&#189;	&frac12;	½
Fraction three-fourths	&#190;	&frac34;	¾
Inverted question mark	&#191;	&iquest;	¿
Capital A, grave accent	&#192;	&Agrave;	À
Capital A, acute accent	&#193;	&Aacute;	Á
Capital A, circumflex accent	&#194;	&Acirc;	Â
Capital A, tilde	&#195;	&Atilde;	Ã
Capital A, diaeresis or umlaut mark	&#196;	&Auml;	Ä
Capital A, ring	&#197;	&Aring;	Å
Capital AE diphthong (ligature)	&#198;	&AElig;	Æ

Description	Decimal-Based Code Value	HTML Entity	How Character Appears on Web Page
Capital C, cedilla	&#199;	&Ccedil;	Ç
Capital E, grave accent	&#200;	&Egrave;	È
Capital E, acute accent	&#201;	&Eacute;	É
Capital E, circumflex accent	&#202;	&Ecirc;	Ê
Capital E, diaeresis or umlaut mark	&#203;	&Euml;	Ë
Capital I, grave accent	&#204;	&Igrave;	Ì
Capital I, acute accent	&#205;	&Iacute;	Í
Capital I, circumflex accent	&#206;	&Icirc;	Î
Capital I, diaeresis or umlaut mark	&#207;	&Iuml;	Ï
Capital Eth, Icelandic	&#208;	&ETH;	Ð
Capital N, tilde	&#209;	&Ntilde;	Ñ
Capital O, grave accent	&#210;	&Ograve;	Ò
Capital O, acute accent	&#211;	&Oacute;	Ó
Capital O, circumflex accent	&#212;	&Ocirc;	Ô
Capital O, tilde	&#213;	&Otilde;	Õ
Capital O, diaeresis or umlaut mark	&#214;	&Ouml;	Ö
Multiplication sign	&#215;	×	×
Capital O, slash	&#216;	&Oslash;	Ø
Capital U, grave accent	&#217;	&Ugrave;	Ù
Capital U, acute accent	&#218;	&Uacute;	Ú
Capital U, circumflex accent	&#219;	&Ucirc;	Û
Capital U, diaeresis or umlaut mark	&#220;	&Uuml;	Ü
Capital Y, acute accent	&#221;	&Yacute;	Ý
Capital THORN, Icelandic	&#222;	&THORN;	Þ
Small sharp s, German (sz ligature)	&#223;	&szlig;	ß
Small a, grave accent	&#224;	&agrave;	à
Small a, acute accent	&#225;	&aacute;	á
Small a, circumflex accent	&#226;	&acirc;	â
Small a, tilde	&#227;	&atilde;	ã

continued

TABLE 15-2 *(continued)*

Description	Decimal-Based Code Value	HTML Entity	How Character Appears on Web Page
Small a, diaeresis or umlaut mark	ä	ä	ä
Small a, ring	å	å	å
Small ae diphthong (ligature)	æ	æ	æ
Small c, cedilla	ç	ç	ç
Small e, grave accent	è	è	è
Small e, acute accent	é	é	é
Small e, circumflex accent	ê	ê	ê
Small e, diaeresis or umlaut mark	ë	ē ë	ë
Small i, grave accent	ì	ì	ì
Small i, acute accent	í	í	í
Small i, circumflex accent	î	î	î
Small i, diaeresis or umlaut mark	ï	ï	ï
Small eth, Icelandic	ð	ð	ð
Small n, tilde	ñ	ñ	ñ
Small o, grave accent	ò	ò	ò
Small o, acute accent	ó	ó	ó
Small o, circumflex accent	ô	ô	ô
Small o, tilde	õ	õ	õ
Small o, diaeresis or umlaut mark	ö	ö	ö
Division sign	÷	÷	÷
Small o, slash	ø	ø	ø
Small u, grave accent	ù	ù	ù
Small u, acute accent	ú	ú	ú
Small u, circumflex accent	û	û	û
Small u, diaeresis or umlaut mark	ü	ü	ü
Small y, acute accent	ý	ý	ý
Small thorn, Icelandic	þ	þ	þ
Small y, diaeresis or umlaut mark	ÿ	ÿ	ÿ
Trademark symbol	™	™	™

Latin Extended-A (U + 0100 - U + 017F)

Once you extend coding past Latin-1 Supplement in Unicode, you begin to veer away from ISO-8859-1 as well. There are specific ISO encodings for different Latin languages. You can find the names of these encodings at `www.alanwood.net/unicode/latin_extended_a.html`.

Alternately, you can simply guarantee the incorporation of these encodings by using UTF-8. The characters in this Unicode block are used in the following languages (among others):

- Afrikaans
- Basque
- Breton
- Catalan
- Croatian
- Czech
- Esperanto
- Estonian
- French
- Frisian
- Greenlandic
- Hungarian
- Latin
- Latvian
- Lithuanian
- Maltese
- Polish
- Provencal
- Rhaeto-Romanic
- Romanian
- Romany
- Sami
- Slovak
- Slovenian
- Sorbian
- Turkish
- Welsh

Latin Extended-B and Latin Extended Additional

The characters in this block are used to write additional languages and to extend Latin encodings. These characters include seldom used characters such as the bilabial click. By the time you reach this territory of encoding, you should definitely be using UTF-8.

Tip

It might seem that encoding your documents in UTF-8 is the best bet. However, mixing encodings is a bad idea and you should consider using a specific codepage for each region — ISO-8859-1 for a largely Western language audience, for example. ■

Summary

This chapter introduced you to the concept of localization of your site. It also covered the basics of Unicode and the different code pages you are likely to encounter when coding your documents for an international audience.

The next two chapters, 16 and 17, cover JavaScript and dynamic HTML.

CHAPTER 16

Scripts

Standard HTML was designed to provide static, text-only documents. No innate intelligence is built into plain HTML, but it is desired, especially in more complex documents or documents designed to be interactive. Enter scripts — svelte programming languages designed to accomplish simple tasks while adhering to the basic premise of the Web, easily deployable content that can play nicely with plain-text HTML.

This chapter covers scripting basics and then provides the details of how to use client-side scripting in your documents.

IN THIS CHAPTER

- Client-Side versus Server-Side Scripting
- Setting the Default Scripting Language
- Including a Script
- Calling an External Script
- Triggering Scripts with Events
- Hiding Scripts from Older Browsers

Client-Side versus Server-Side Scripting

There are two basic varieties of scripting, *client-side* and *server-side*. As their names imply, the main difference is where the scripts are actually executed.

Client-side scripting

Client-side scripts are run by the client software — that is, the user agent. As such, they impose no additional load on the server, but the client must support the scripting language being used.

JavaScript is the most popular client-side scripting language, but JScript and VBScript are also widely used. Client-side scripts are typically embedded in HTML documents and deployed to the client. Client users can usually easily view these scripts.

For security reasons, client-side scripts generally cannot read or write to the server or client file system.

Server-side scripting

Server-side scripts are run by the Web server. Typically, these scripts are referred to as CGI scripts, CGI being an abbreviation for Common Gateway Interface, the first interface for server-side Web scripting. Server-side scripts impose more of a load on the server, but generally don't influence the client — even output to the client is optional. The client may have no idea that the server is running a script.

Perl, Python, PHP, Java, ASP, and ASP.NET are all examples of server-side scripting languages. The script typically resides only on the server, but is called by code in the HTML document.

Although server-side scripts cannot read or write to the client's file system, they usually have some access to the server's file system. Therefore, it is important that the system administrator take appropriate measures to secure server-side scripts and limit their access.

Note

Unless you are a system administrator on the Web server you use to deploy your content, your ability to use server-side scripts is probably limited. Your Internet service provider (ISP) or system administrator has policies that allow or disallow server-side scripting in various languages and performing various tasks.

If you intend to use server-side scripts, you should check with your ISP or system administrator to determine what resources are available to you. ■

This chapter deals only with client-side scripting.

Setting the Default Scripting Language

To embed a client-side script in your document, you use the script (`<script>`) tag. This tag has the following, minimal format:

```
<script type="script_type">
```

The value of `script_type` depends on the scripting language you are using. The following are generally used script types:

- `text/ecmascript`
- `text/javascript`
- `text/jscript`
- `text/vbscript`
- `text/vbs`
- `text/xml`

For example, if you are using JavaScript, your script tag would resemble the following:

```
<script type="text/javascript">
```

Note

The W3C recommends that you specify the default script type you are using in an appropriate `META` tag in your document. Such a tag resembles the following:

```
<META http-equiv="content_script_type"
content="text/javascript">
```

This does not obviate the need for the `type` attribute in each `script` tag. You must still specify each `script` tag's type for your documents to validate against HTML 4.01. ■

If your script is encoded in a character set other than the one used for the rest of the document, you should also use the `charset` attribute to specify the script's encoding. This attribute has the same format as the `charset` attribute for other tags:

```
charset="character_encoding_type"
```

Including a Script

To include a script in your document, place the script's code within `<script>` tags. For example, consider the following script:

```
<script type="text/javascript">
  function NewWindow(url){
    var fin=window.open(url,"","width=800,height=600,
    scrollbars=yes,resizable=yes");
}
</script>
```

You can include as much scripting code between the tags as needed, providing that the script is syntactically sound. Scripts can be included within a document's head or body sections, and you can include as many script sections as you like. Note, however, that nested `script` tags are not valid HTML.

Generally, you will want to place your scripts in the head section of your document so the scripts are available as the rest of the page loads. You may occasionally want to embed a script in a particular location in the document — in those cases, place an appropriate `script` block in the body of the document. For example, you may want a script in close proximity to a paragraph it affects. In that case, you would place it in-line, as shown in the following example:

```
<h2>Spa Services</h2>
<p>The Oasis of Tranquility offers a full menu of services to
renew the real you that lies within. Begin in one of our two
relaxation centers, then enjoy an invigorating body and facial care,
deep soothing massage therapies, and a host of other indulgent
treatments that pamper you on the outside, and revive you from
within. In addition to our many spa services, take a refreshing dip
in the swimming pool, melt in one of our whirlpool spas, or
rejuvenate in the sauna.</p>
```

```
<script type="text/javascript">
   ... script contents go here ...
</script>
<h2>Give the Gift of Tranquility</h2>
<p>All services at the Oasis of Tranquility can be experienced
individually, or selected a la carte to create you own personalized
day of pampering.  Gift certificates are excellent for surprising
your loved ones with an hour or a day of pampering and
rejuvenation.</p>
```

Calling an External Script

If you have some scripts that you want to use in multiple documents, consider placing them in an external file. You can then use the `script` tag's `src` attribute to specify that the script content can be found in that file. For example, suppose you want to include the following script in multiple documents:

```
function NewWindow(url){
  var fin=window.open(url,"","width=800,height=600,
  scrollbars=yes,resizable=yes");
}
```

You can place the script in a text file on the server and specify the file's URL in the appropriate `script` tag's `src` attribute. Suppose the preceding file were stored in the file `scripts.js` on the server. Your script tag would then resemble the following:

```
<script type="text/javascript" src="scripts.js"></script>
```

Note that even though the script element's body is empty, it still requires the closing `</script>` tag.

One major advantage to external script files is that if you need to edit the script, you can edit it in one place — the external file — and the change is effected in all the documents that include it.

Triggering Scripts with Events

Most HTML tags can include event attributes that can be used to trigger scripts. Table 16-1 lists these attributes and their use for triggering scripts.

Cross-Ref

See Appendix A for a comprehensive list of what tags support event attributes. ■

Note

Many of the event attribute triggers are dependent on the element(s) to which they apply being "in focus" at the time of the trigger. ■

TABLE 16-1

Event Attributes

Attribute	Trigger Use
onclick	When item enclosed in the tag is clicked
ondblclick	When item enclosed in the tag is double-clicked
onmousedown	When mouse button is pressed while mouse pointer is over the item enclosed in the tag
onmouseup	When mouse button is released while mouse pointer is over item enclosed in the tag
onmouseover	When mouse pointer is placed over the item enclosed in the tag
onmousemove	When mouse is moved within the item enclosed in the tag
onmouseout	When mouse is moved outside of the item enclosed in the tag
onblur	When item enclosed in the tag has focus removed
onfocus	When item enclosed in the tag receives focus
onload	When the document finishes loading (valid only in the <body> tag)
onunload	When the document is unloaded — when the user navigates to another document (valid only in the <body> tag). This event is often used to create pop-ups when a user leaves a site.
onsubmit	When a form has its Submit button pressed (valid only in <form> tags)
onreset	When a form has its Reset button pressed (valid only in <form> tags)
onkeypress	When a key is pressed while the mouse pointer is over the item enclosed in the tag
onkeydown	When a key is pressed down while the mouse pointer is over the item enclosed in the tag
onkeyup	When a key is released while the mouse pointer is over the item enclosed in the tag

Event triggers have a variety of uses, including the following:

- **Form data verification** — Using the onchange and onselect attributes, you can verify each field as it is changed or selected. Using onsubmit and onreset, you can verify or reset an entire form when the appropriate button is clicked.
- **Image animation** — Using the onmouseover and onmouseout attributes, you can animate an image when the mouse pointer passes over it.
- **Mouse navigation** — Using the onclick and ondblclick attributes, you can trigger user agent navigation when a user clicks or double-clicks an element.

For example, you can create images to use as buttons on your page. Figure 16-1 shows two images for use on a button. The image on the left is used for general display, while the image on the right is used when the mouse is over the button.

FIGURE 16-1

Two images for use as a button

Tip

Users appreciate visible feedback from active elements on your page. As such, it is important to always provide visible changes to navigation elements. Links should have a visibly different style when moused over, as should navigation buttons. ■

Combining `onmouseover`, `onmouseout`, and `onclick` events, you can easily create a button that reacts when the mouse is over it and navigates to a new page when clicked. Consider the following document, which uses a few JavaScript scripts and events to create a navigation button:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<META http-equiv="Content-Script-Type"
      content="text/javascript">
```

```
<title>Event Buttons</title>
<script type="text/javascript">
  // Get the specified object by ID
  //  (browser specific method used)
  function getObj(id) {
    if (document.getElementById) {
      this.obj = document.getElementById(id);
}     else if (document.all) {
      this.obj = document.all[id];
}     else if (document.layers) {
      this.obj = document.layers[id];
}
}
   // Activate the specified button
   function activate(bname) {
     imageid = bname + "button";
     iname = bname + "On.jpg";
     x = new getObj(imageid);
     x.obj.src = iname;
}
   // Deactivate the specified button
   function deactivate(bname) {
     imageid = bname + "button";
     iname = bname + "Off.jpg";
     x = new getObj(imageid);
     x.obj.src = iname;
}

</script>
</head>
<body>
<p>
<img alt="Home page" id="homebutton"
  src="HomeOff.jpg"
  onmouseover="activate('home')"
  onmouseout="deactivate('home')"
  onclick="document.location='home.html"'
/>
</p>
</body>
</html>
```

When the document loads, the button is displayed in its inactive (off) state, as shown in Figure 16-2. When the mouse is placed over the button, the `onmouseover` event launches the JavaScript `activate` function, which changes the image's `src` attribute, causing the button to be displayed as active (on), as shown in Figure 16-3.

When the mouse leaves the button, the `onmouseout` event launches the `deactivate` function, returning the button display to its inactive display state by changing the `src` attribute back to the original image. When the button is clicked, the `onclick` event changes the `location` property of the user agent, effectively navigating to a new page (in this case `home.html`).

Note that the JavaScript code for the `onclick` attribute is contained directly in the value of the attribute — because the code is short, placing it in-line within the attribute is more economical than creating a separate function.

FIGURE 16-2

The button is initially displayed in its inactive (off) state.

Tip

You can place several lines of code within the value section of the event attributes. Just ensure that you end each line (where appropriate) with a semicolon. ■

This example only scratches the surface of what JavaScript can do within an HTML document. Similar methods can be used to manipulate form objects and other elements within your documents.

Cross-Ref

Additional methods and examples are covered in Chapter 17. ■

FIGURE 16-3

The button is changed to active (on) when the mouse is over it.

Hiding Scripts from Older Browsers

Not all user agents support JavaScript. Many of the older user agents are not JavaScript-enabled, and some of the latest user agents may have JavaScript disabled by their users.

Note

Most modern browsers will ignore scripts of types they do not recognize. ■

If you are concerned about older browsers not recognizing your scripts, you should *hide* your scripts so that older browsers will ignore them (instead of trying to render them).

To hide your scripts, simply place them within a special set of comment tags. The only difference between normal comment tags and script-hiding tags is that the closing tag contains two slashes (//). Those two slashes enable browsers that support scripting to find the script.

For example, the following structure will effectively hide the scripts within the <script> tag:

```
<script type="text/javascript">
  <!-- hide scripts from older browsers
  --- Script content ---
  // -->
</script>
```

Summary

This chapter introduced how to add basic intelligence and dynamic content to your site via client-side scripting. You learned how to embed scripts in your documents and how to utilize external script files. The chapter also covered the use of event attributes to trigger scripts from user actions.

The next chapter covers Dynamic HTML, which enables you to influence a document's content using scripting, and shows how to put the basic knowledge of scripting you learned here to maximum use.

CHAPTER 17

Dynamic HTML

Dynamic HTML (DHTML) is a combination of HTML, CSS, and JavaScript, used to create dynamic Web page effects. These can be animations, dynamic menus, text effects such as drop shadows, text that appears when a user rolls over an item, and other similar effects.

This chapter introduces DHTML by reviewing some JavaScript basics and providing a look at the Document Object Model (DOM), which enables you to access HTML elements so you can change their properties and/or content. Examples of common DHTML techniques are provided.

Note

In a very strict, technical sense, DHTML is thought of as containing code that is targeted toward level 4 browser architecture with a lot of proprietary code. For example, such a script would be written for a particular platform, use proprietary hooks and code existing only on that platform, and would be incompatible with other platforms.

However, Document Object Model (DOM) scripting has emerged to enable scripts that follow cross-browser–compatible standards, and hence are more compatible with more platforms.

That said, DHTML is still the predominant term used for the dynamic combination of HTML, CSS, and JavaScript, and as such is used here. ■

IN THIS CHAPTER

- The Need for DHTML
- How DHTML Works
- The Document Object Model
- The JavaScript DOM
- Using Event Handlers
- Accessing an Element by Its ID
- Cross-Browser Compatibility Issues
- DHTML Examples
- Form Automation: Check boxes

The Need for DHTML

DHTML, when used correctly, can significantly enhance the user experience. DHTML was originally best known for its flashy effects. These still exist, but their importance is questionable, and when used improperly they can be

annoying for your users. Fancy text animations and bouncing balls might be fun to write, but they're not so much fun for the user. This chapter focuses on the more practical aspects of DHTML, most of which are related to navigation. After all, your website should be all about the user experience.

Tip

Whenever you create an enhancement to your website, you should always ask, "Does this improve the user experience? Can they navigate my site more easily? Read my Web page more easily?" If the answer to any of these questions is no, rethink the enhancement. ■

How DHTML Works

DHTML can work either by applying certain CSS properties or by using JavaScript to directly manipulate HTML elements. When using JavaScript, DHTML takes advantage of a browser's object model, which is a tree of objects based on the element set of HTML and the property set of CSS. When you code against that object model, you can change an element's properties, which are associated with an element's attributes. An element's attributes, in fact, are referred to as *properties* in a JavaScript environment. How these properties are referred to, and what actions (methods) you can take on them, is determined by the DOM.

Actually, several DOMs are available for your scripting needs. However, only two are pertinent for typical websites — the pure object DOM created by the World Wide Web Consortium (W3C) and the JavaScript DOM consisting of JavaScript methods mapped to document objects. The following sections cover both of these models.

The Document Object Model

Most Web developers are familiar with the concept of DHTML and the underlying DOMs developed by Netscape and Microsoft for their respective browsers. However, a unifying DOM developed by the W3C is even more powerful, because of its compatibility, and is much more popular with more professional developers. The W3C DOM has several advantages over the DHTML DOMs — using its node structure, it is possible to easily navigate and change documents despite the user agent employed to display them. This chapter covers the basics of the W3C DOM and explains how to use JavaScript to manipulate it.

Note

The W3C DOM is much more complex than shown within this chapter. Several additional methods and properties are at your disposal to use in manipulating documents, many more than we have room to address in this chapter. Further reading and information on the standard can be found on the W3C site at `www.w3.org/TR/2000/WD-DOM-Level-1-20000929/Overview.html`. ■

The history of the DOM

The DOM was developed by the W3C to allow programming languages access to the underlying structure of a Web document. Using the DOM, a program can access any element in the document, determining and changing attributes and even removing, adding, or rearranging elements at will.

It's important to note that the DOM is a type of application program interface (API), allowing any programming language access to the structure of a Web document. The main advantage of using the DOM is the capability to manipulate a document without another trip to the document's server. As such, the DOM is typically accessed and used by client-side technologies, such as JavaScript.

The first DOM specification (Level 0) was developed at the same time as JavaScript and early browsers. It is supported by Netscape 2 onward.

Two intermediate DOMs were supported by Netscape 4 onward and Microsoft Internet Explorer (IE) versions 4 and 5 onward. These DOMs were proprietary to the two sides of the browser coin — Netscape and Microsoft IE. The former used a collection of elements referenced through a `document.layers` object, whereas the latter used a `document.all` object. To be truly cross-browser compatible, a script should endeavor to cover both of these DOMs instead of only one or the other.

The latest, well-supported DOM specification (Level 2) is supported by Mozilla and Microsoft Internet Explorer version 5 onward. Both browser developers participated in the creation of this level of the DOM. However, Microsoft chose to continue to support its `document.all` model as well, while Netscape discontinued its `document.layers` model.

Keep in mind that because the DOM was originally intended to allow programs to navigate and change XML, not HTML, documents, it contains many features a Web developer dealing only with HTML may never need.

Understanding the DOM

The basis of the DOM is to recognize each element of the document as a node connected to other nodes in the document and to the document root itself. The best way to understand the structure is to look at an example. The following code shows a document that renders as shown in Figure 17-1, and whose DOM is illustrated in Figure 17-2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Sample DOM Document</title>
<style type="text/css">
  div.div1 { background-color: #999999; }
```

```
    div.div2 { background-color: #BBBBBB; }
    table, table * { border: thin solid black; }
    table { border-collapse: collapse; }
    td { padding: 5px; }
</style>
<script type="text/JavaScript">
</script>
</head>
<body>
<div class="div1">
  <h1>Heading 1</h1>
  <table>
    <tr><td>Cell 1</td><td>Cell 2</td></tr>
    <tr><td>Cell 3</td><td>Cell 4</td></tr>
  </table>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam <b>nonummy nibh euismod</b> tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>
<div class="div2">
  <h1>Heading 2</h1>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam nonummy nibh euismod tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
  <ol id="sortme">An ordered list
    <li>Gamma</li>
    <li>Alpha</li>
    <li>Beta</li>
  </ol>
</div>
</body>
</html>
```

As you can see, each node is joined to its neighbors using a familiar parent/child/sibling relationship. For example, the first `div` node is a child of the `body` node, and the `div` node in turn has three children: an `h1` node, a `P` node, and an `ol` node. Those three children (`h1`, `p`, and `ol`) have a sibling relationship to one another.

Plain text and usually the content of nodes such as paragraphs (`p`) are referenced as textual nodes and are broken down, as necessary, to incorporate additional nodes. This can be seen in the first `p` node, which contains a bold (`b`) element. The children of the `P` node include the first bit of text up to the bold element, the bold element, and the text after the bold element. The bold element (`b`) in turn contains a text child, which contains the bolded text.

The relationships between nodes can be explored and traversed using the DOM JavaScript bindings, as described in the next section.

FIGURE 17-1

A sample document

DOM node properties and methods

The W3C DOM includes several JavaScript bindings that can be used to navigate a document's DOM. A subset of those bindings, used in JavaScript as properties and methods, is listed in Tables 17-1 and 17-2. The first table describes JavaScript's properties.

Note

A full list of DOM Level 1 JavaScript bindings can be found on the W3C's Document Object Model Level 1 pages, at `www.w3.org/TR/2000/WD-DOM-Level-1-20000929/ecma-script-language-binding.html`. ■

Note

A full list of DOM Level 2 JavaScript bindings can be found on the W3C's Document Object Model Level 2 pages, at `www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/ecma-script-binding.html`. ■

FIGURE 17-2

Diagram of the sample document's DOM

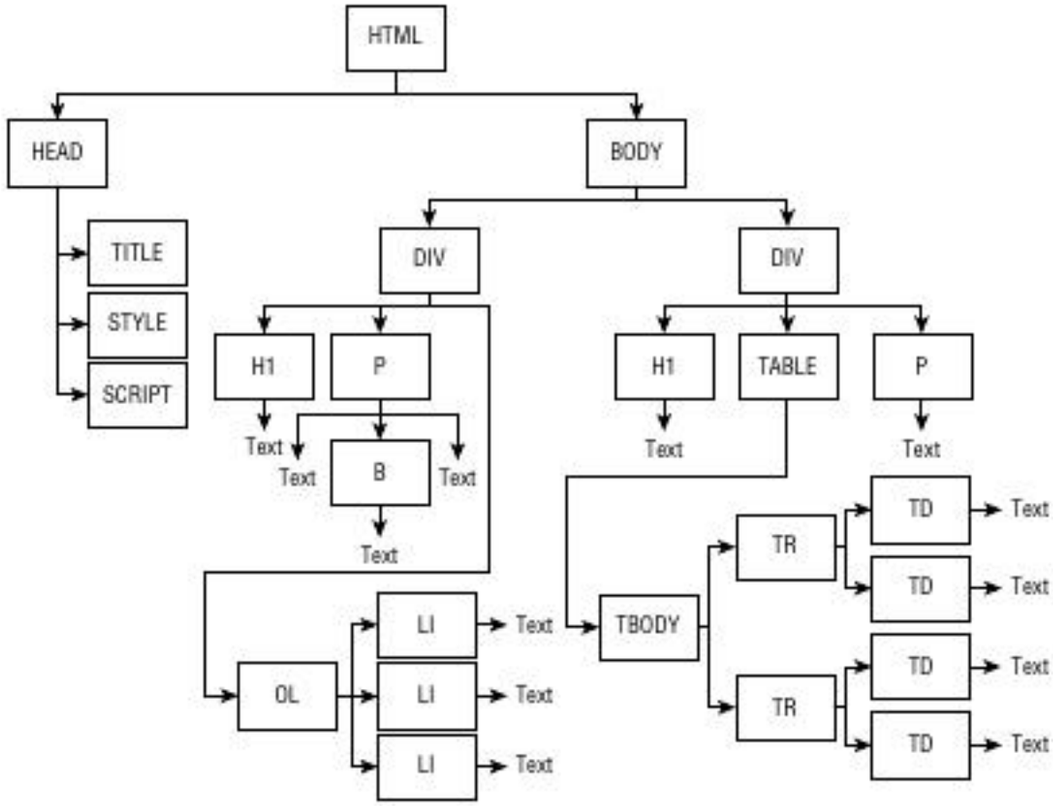


TABLE 17-1

JavaScript DOM Property Bindings

Property	Description
`attributes`	This read-only property returns a `NamedNodeMap` containing the specified node's attributes.
`childNodes`	This read-only property returns a node list containing all the children of the specified node.
`firstChild`	This read-only property returns the first child node of the specified node.
`lastChild`	This read-only property returns the last child node of the specified node.
`nextSibling`	This read-only property returns the next sibling of the specified node.
`nodeName`	This read-only property returns a string containing the name of the node, which is typically the name of the element (`p`, `div`, `table`, and so on).
`nodeType`	This read-only property returns a number corresponding to the node type (1 = element, 2 = text).

Property	Description
nodeValue	This property returns a string containing the contents of the node and is only valid for text nodes.
ownerDocument	This read-only property returns the root document node object of the specified node.
parentNode	This read-only property returns the parent node of the specified node.
previousSibling	This read-only property returns the previous sibling of the specified node. If there is no node, then the property returns null.

TABLE 17-2

JavaScript DOM Method Bindings

Method	Description
appendChild(newChild)	Given a node, this method inserts the newChild node at the end of the children and returns a node.
cloneNode(deep)	This method clones the node object. The parameter deep — (a Boolean) — specifies whether the clone should include the source object's attributes and children. The return value is the cloned node(s).
createElement(element)	This method creates an HTML element of the specified type.
createTextNode("text")	This method creates a new text node using the specified text.
getAttribute(attrib)	This method returns the value of the specified attribute.
getElementById(id)	This method returns a reference to the element having the id specified.
getElementByTagName (element)	This method returns the number of a specified element found in the document.
hasChildNodes()	This method returns true if the node object has children nodes, false if the node object has no children nodes.
insertBefore(newChild, refChild)	Given two nodes, this method inserts the newChild node before the specified refChild node and returns a node object.
removeChild(oldChild)	Given a node, this method removes the oldChild node from the DOM and returns a node object containing the node removed.

continued

TABLE 17-2 *(continued)*

Method	Description
`replaceChild(newChild, oldChild)`	Given two nodes, this method replaces the `oldChild` node with the `newChild` node and returns a node object. Note that if the `newChild` is already in the DOM, it is removed from its current location to replace the `oldChild`.
`setAttribute(attribute, value)`	This method sets the specified `attribute` to the specified `value`.

Traversing and changing a document's nodes

Using the bindings from the preceding section, it is possible to write JavaScript to navigate through a document using nodes and change node attributes. Remember that nodes typically correspond to HTML elements, so changing nodes changes the document's HTML.

The following code includes a recursive JavaScript function, `findNode()`, that looks at each node and child node in a document, searching for the node that is an `ol` element with an `id` of `sortme`. The comments in the code outline how the function operates:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>DOM Find Node</title>
<style type="text/css">
  div.div1 { background-color: #999999; }
  div.div2 { background-color: #BBBBBB; }
  table, table * { border: thin solid black; }
  table { border-collapse: collapse; }
  td { padding: 5px; }
</style>
<script type="text/JavaScript">
// Starting at node "startnode," transverse the document
//   looking for an element named "nodename" with an id
//   of "nodeid"
//////////////////////////////////////////////////////
function findNode(startnode,nodename,nodeid) {
  var foundNode = false;
  // Check if our starting node is what we are looking for
  if ( startnode.nodeName == nodename &&
       startnode.id == nodeid ) {
    foundNode = startnode;
  // If startnode is not what we are searching for
}   else {
    look_thru_children:
    // If current startnode has children
```

```
    if ( startnode.hasChildNodes() ) {
      var children = startnode.childNodes;
      // Look through each child and its children
      //   (by recursing through this function)
      for (var i = 0; i < children.length; i++) {
        foundNode = findNode(children[i],nodename,nodeid);
        // If we find what we are looking for, stop recursion
        if (foundNode) { break look_thru_children;}
}
}
}
  // Return the node
  return foundNode;
}
////////////////////////////////////////////////////
// Kick off the search (runs from <body> onload)
function dofind() {
  alert("Click OK to find 'sortme' node");
  var node = findNode(document,"OL","sortme");
  alert("Found node: " + node.nodeName);
}
</script>
</head>
<body onload="dofind()">
<div class="div1">
  <h1>Heading 1</h1>
  <table>
    <tr><td>Cell 1</td><td>Cell 2</td></tr>
    <tr><td>Cell 3</td><td>Cell 4</td></tr>
  </table>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam <b>nonummy nibh euismod</b> tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
</div>
<div class="div2">
  <h1>Heading 2</h1>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
  elit, sed diam nonummy nibh euismod tincidunt ut laoreet
  dolore magna aliquam erat volutpat. Ut wisi enim ad minim
  veniam, quis nostrud exerci tation ullamcorper suscipit
  lobortis nisl ut aliquip ex ea commodo consequat.</p>
  <ol id="sortme">An ordered list
    <li>Gamma</li>
    <li>Alpha</li>
    <li>Beta</li>
  </ol>
</div>
</body>
</html>
```

The script opens an alert window displaying the found node's name.

Tip

The DOM provides another, easier mechanism to find an element with a particular `id` — namely, the `getElementById()` method of the document object. In fact, the entire search function in the preceding script can be replaced with one line:

```
node = document.getElementById("sortme");
```

The previous method of traversing the DOM was used only to illustrate how you can manually navigate and search the DOM, if necessary. ■

Just as you can navigate downward through the document using the `childNodes` method, you can also navigate across the DOM with `previousSibling` or `nextSibling` (selecting adjacent siblings of a particular node) or up the DOM using `parentNode`.

You can also use the JavaScript bindings to change a node's value. For example, suppose you have a paragraph element with an ID of `"edit"` similar to this:

```
<p id="edit"> ... </p>
```

You can change the text within the element using the following JavaScript code:

```
// Find the element, assign it to "node"
var node = document.getElementById("edit");
  // Make sure the node is text (nodeType = 3)
  if (node.firstChild.nodeType == 3) {
    // Change the text to "Changed text"
    node.firstChild.nodeValue = "Changed text";
}
```

You can also copy one element's text to another, using code similar to the following:

```
node2.nodeValue = node1.nodeValue;
```

Ultimately, you can copy an entire node to another, using code similar to this:

```
// Copy a node and all of its properties to another
node2 = node1;
```

The JavaScript DOM

The standardized form of JavaScript is called ECMAScript. This is a relevant fact because, usually, if you confine your scripting to the conventions of the W3C's Level 1 DOM and ECMAScript, you'll be pretty successful at achieving cross-browser scripting compatibility.

Note

You can find the specification for ECMAScript at `www.ecma-international.org/publications/standards/Ecma-262.htm`. ■

The W3C's Level 1 Core DOM is basically a set of properties and methods that can be accessed from a given element. For example, one of the most ubiquitous (and dastardly, in many people's opinion) methods is the `window.open()` method, which makes it possible for JavaScript to open a new browser window in which advertising pop-ups appear the majority of the time. The `open()` method acts on the `window` object, which, although not an element (the DOM isn't restricted to elements), is still an object that can be manipulated by script.

JavaScript has a host of built-in objects that can be used to access the user agent and the document it contains. This section introduces the various objects and how JavaScript can use them. Figure 17-3 shows the ECMAScript (JavaScript) Core DOM, consisting of the various objects, properties, and methods to access document objects. The sections that follow provide more detail on the DOM's elements.

Note

Use of the JavaScript DOM is a stark contrast to using the W3C DOMs. The former has a host of built-in objects that allow you to directly access objects in a document, whereas the latter utilizes a set of standard methods for accessing and manipulating elements as nodes. Generally, use of the JavaScript DOM is easier and more straightforward, but it does require more advanced knowledge of a document's layout and contents, whereas the W3C DOM tools can act upon more abstract documents. ■

Tip

For quick-and-dirty scripts, stick with the JavaScript DOM. For more robust and variable scripting, consider the W3C DOM. ■

The window object

The `window` object is the top-level object for an XHTML document. It includes properties and methods to manipulate the user agent window. The `window` object is also the top-level object for most other objects.

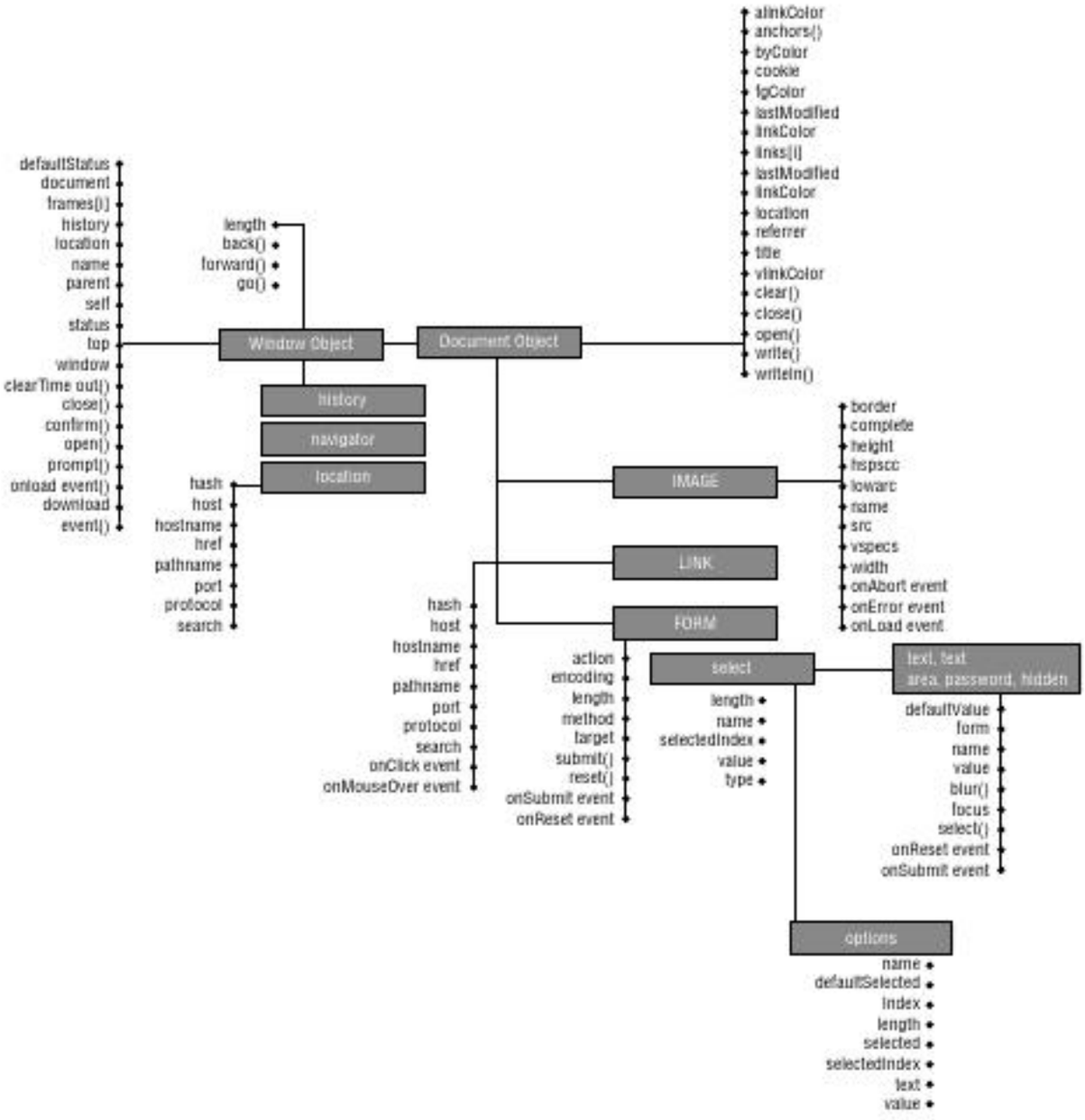
Using the `window` object, not only can you work with the current user agent window, you can also open and work with new windows. The following code will open a new window displaying a specific document:

```
NewWin = window.open("example.htm","newWindow",
  "width=400,height=400,scrollbars=no,resizable=no");
```

The `open` method takes three arguments: the URL of the document to open in the window, the name of the new window, and options for the window. For example, the preceding code opens a window named `newWindow` containing the document `example.htm` and will be 400 pixels square, be non-resizable, and have no scroll bars.

FIGURE 17-3

The Core DOM used by ECMAScript (JavaScript)



The options supported by the `open` method include the following:

- `toolbar = yes|no` — Controls whether the new window will have a toolbar
- `location = yes|no` — Controls whether the new window will have an address bar
- `status = yes|no` — Controls whether the new window will have a status bar
- `menubar = yes|no` — Controls whether the new window will have a menu bar
- `resizeable = yes|no` — Controls whether the user can resize the new window

- `scrollbars - yes|no` — Controls whether the new window will have scrollbars
- `width - pixels` — Controls the width of the new window
- `height - pixels` — Controls the height of the new window

Note
Not all user agents support all options. ■

The `window` object can also be used to size and move a user agent window. One interesting DHTML effect is to shake the current window. The following function can be used to cause the user agent window to visibly shudder:

```
function shudder() {
  // Move the document window up and down 5 times
  for (var i=1; i<= 5; i++) {
    window.moveBy(8,8);
    window.moveBy(-8,-8);
}
}
```

You can use other methods to scroll a window (`scroll`, `scrollBy`, `scrollTo`) and to resize a window (`resizeBy`, `resizeTo`).

The document object

You can use the JavaScript `document` object to access and manipulate the current document in the user agent window. Many of the collection objects (`form`, `image`, and so on) are children of the `document` object.

The `document` object supports a `write` and `writeln` method, both of which can be used to write content to the current document. For example, the following code results in the current date being displayed (in mm/dd/yyyy format) wherever the code is inserted in the document:

```
<script type="text/JavaScript">
  today = new Date();
  document.write((today.getMonth()+1) + "/" + today.getDate() +
    "/" + today.getFullYear());
</script>
```

The `open` and `close` methods can be used to open and then close a document for writing. Building on the examples in the earlier section "The window object," the following code can be used to spawn a new document window and write the current date to the new window:

```
<script type="text/JavaScript">
  today = new Date();
  newWin = window.open("","","width=400,height=400,
     scrollbars=no,resizable=no");
```

```
  newDoc = newWin.document.open();
  newDoc.write((today.getMonth()+1) + "/" + today.getDate() +
    "/" + today.getFullYear());
  newDoc.close();
</script>
```

The form object

You can use the `form` object to access form elements in a document. The `form` object supports `length` and `elements` properties — the former property returns how many elements (fields) are in the form; the latter contains an array of form element objects, one per field. You can also access the form elements by their `name` attribute. For example, the following code will set the value of the `addlength` field to the length of the address field using the form name and element names to address the various values:

```
...
<head>
<script type="text/JavaScript">
  function dolength() {
    document.form1.addlength.value =
      document.form1.address.value.length;
}
</script>
</head>
<body>
<p>
<form name="form1" action="handler.cgi" method="post">
Length: <input type="text" name="addlength" size="5" /><br />
Address: <input type="text" name="address" size="30"
          onkeyup="dolength();" />
</form>
</p>
...
```

The `form` object can be used for a variety of form automation techniques.

The location object

The `location` object can be used to manipulate the URL information about the current document in the user agent. Various properties of the `location` object are used to store individual pieces of the document's URL (protocol, hostname, port, and so on). For example, you could use the following code to piece the URL back together:

```
with (document.location) {
  var url = protocol + "//";
  url += hostname;
  if (port) { url += ":" + port;}
  url += pathname;
```

```
  if (hash) { url += hash;}
}
```

The preceding example is shown only to illustrate how the various pieces relate to one another; the `location.href` property contains the full URL.

One popular method of using the `location` object is to cause the user agent to load a new page. To do so, your script simply has to set the `document.location` object to the desired URL. For example, the following code will cause the user agent to load the Yahoo.com home page:

```
document.location = "http://www.yahoo.com";
```

The history object

The `history` object is tied to the history function of the user agent. Using the `history` object, your script can navigate up and down the history list. For example, the following code acts as though the user used the browser's back feature, causing the user agent to load the previous document in the history list:

```
history.back();
```

Other properties and methods of the `history` object allow more control over the history list. For example, the `history.length` property can be used to determine the number of entries in the history list.

The self object

You can use the `self` object to refer to an element making the function call. This object is typically used when calling JavaScript functions, allowing the function to operate on the object initiating the call. For example, the following code passes a reference to the button to the `dosomething()` function:

```
<input type="button" value="Click Me" id="button"
onclick="dosomething(self);" />
```

The function can then use that reference to operate on the object that initiated the call:

```
function dosomething(el) {
... // do something with the element referenced by el ...
}
```

For example, the following function can be used to change the color of an element when called with a reference to that element:

```
function changecolorRed(el) {
  el.style.color = "red";
}
```

That function can then be added to an event of any element, as in the following `onclick` event example:

```
<p onclick="changecolorRed(this);">When clicked,
    the text will change to red.</p>
```

Using Event Handlers

Notice the `onclick` attribute in the following code fragment:

```
<div onClick="this.style.fontSize='60px';
this.style.color='red'">
```

This tells the browser that when the user clicks the `div` element something should happen. In this case, that something is that the following two attributes of the style element will change:

- `style.fontSize` tells the browser to change the font size to 60 pixels.
- `style.color` tells the browser to change the color to red.

The two statements are JavaScript code, making use of the JavaScript hook into the document's CSS. The `onClick` attribute is actually an event handler. An event is something that happens, as you probably already know. A party, for example, is an event. When a human triggers the `onparty` event, sometimes that human falls down drunk. When a human triggers an `onClick` event in a browser, more benign things take place, such as text color changes, menu changes, and so on.

Besides placing spurious code in the element, you can also place a function call as a function call to the `onClick` event. For example, if you have a function named "ChangeDiv" defined in a `<script>` section earlier in your document, you could use the following `onClick`:

```
<div onClick="ChangeDiv(this);">
```

All the code to change the `div` element could then be placed in the function and used by multiple elements.

Note

The "this" used in the previous examples is shorthand for referring to the element in which the code was placed, or in this case, the `div`. ■

Table 17-3 shows the common event handlers associated with JavaScript.

When one of these events takes place and the appropriate handler is included in one or more elements, the corresponding code is executed.

Note

Many browsers have their own, custom event handlers, but if you stick with those found in Table 17-3, you'll find cross-compatibility issues much easier to solve. ■

TABLE 17-3

JavaScript Event Handlers

Event Handler	Usage
onAbort	Occurs when a user stops an image from loading.
onBlur	Occurs when a user's mouse loses focus on an object. Focus is when the cursor is active on an object, such as a form input field. When a cursor is clicked within the field, it has focus, and when the mouse is taken out of the field, it loses focus, causing an onBlur event.
onChange	Occurs when a change takes place in the state of an object — for example, when a form field loses focus after a user changes some text within the field.
onClick	Occurs when a user clicks an object.
onError	Occurs when an error occurs in the JavaScript.
onFocus	Occurs when a user's mouse enters a field with a mouse click.
onLoad	Occurs when an object, such as a page (as represented by the body element), is loaded into the browser.
onMouseOut	Occurs when a mouse no longer hovers over an object.
onMouseOver	Occurs when a mouse begins to hover over an object.
onSelect	Occurs when a user selects text.
onSubmit	Occurs when a form is submitted.
onUnload	Occurs when an object is unloaded.

Accessing an Element by Its ID

One of the surest methods to access a document's elements is to use the `getElementById()` function. This function is supported by any DOM Level 1-compliant user agent, so it can be relied upon to access elements that have a properly assigned ID attribute.

The `getElementById()` function's syntax is straightforward:

```
element = document.getElementById("elementID");
```

For example, the following code would assign a reference to the `address` field to the `element` variable:

```
element = document.getElementById("address");
...
<input type="text" size="30" id="address">
```

Once assigned, the `element` variable can be used to access the referenced field's properties and methods:

```
addresslength = element.length;
```

Tip

Before using `getElementById()`, you should test the user agent to ensure that the function is available. The following `if` statement will generally ensure that the user agent supports the appropriate DOM level and thus `getElementById()`:

```
if (document.all || document.getElementById) {
  ...getElementById should be available, use it here...
} ■
```

Cross-Browser Compatibility Issues

The most important caveat to exploring DHTML is that there are a ton of compatibility issues. The newest iterations of Firefox/Mozilla/Netscape and Internet Explorer have actually begun to come closer together, but developers working with DHTML during the height of the browser wars quickly learned that developing cross-browser DHTML was a very difficult proposition. As a result, most large professional sites eschew complex DHTML in favor of simpler cross-browser routines to improve navigation and other facets of the user experience, rather than excessive visual effects.

Browser detection: querying for identification

You can detect what kind of a browser a user is using by running a browser-detection script. This kind of script, along with some more finely tuned type of object detection, described in the next section, is sometimes referred to as *browser sniffing*. At its simplest, a typical browser-detection script looks like this:

```
<script language="JavaScript">
<!--
var bName =navigator.appName;
var bVer = parseFloat(navigator.appVersion);
if (bName == "Netscape")
  var browser = "Netscape Navigator"
else
  var browser = bName;
document.write("You are currently using ", browser, " ",
bVer, ".");
// -->
</script>
```

However, this method is inexact at best. Many browsers report erroneous data in their ID strings, and knowing a browser's name and version doesn't guarantee that it supports particular features. A better method is to test for each key feature you use — or objects that exist to support that feature — as described in the next section.

Browser detection: object detection

Object detection is a more precise way of browser sniffing. It examines a browser's support for various aspects of the object model. This avoids the potential for successfully checking a browser version but not confirming that the browser actually supports a specific object property or method. For this reason, object detection is the preferred method for browser sniffing and is considered a best practice. In addition, unless you have the object model of all the different browsers memorized, it's difficult to know which browser supports which object. It's easier to just check and see if a browser supports a specific object's properties or methods.

The principles used in object detection are quite similar to those used in browser detection. You make use of JavaScript `if` statements to check a browser's support for a named object's properties or methods. If the browser does support the object, you execute some given code. For example, using regular expressions can be very handy in JavaScript, but not if your users' browsers don't support them. So you create a simple detection script to see if they do:

```
if (window.RegExp) {
  // execute some regular expressions
} else {
  // provide an alternative to regular expressions
}
```

DHTML Examples

This section offers a few practical examples of DHTML. The scripts you'll see here are necessarily simple to get you started. You'll find a ton of resources on the Internet for additional help, including a vast array of freely available scripts you can customize for your own use. We'll take a look at a few of the most popular DHTML routines.

Form Automation: Check boxes

Dynamic HTML is very useful when used with `form` elements. By using events to tie specific elements to JavaScript functions, you can perform a wide array of automated tasks.

One popular automation technique is to add a special check box that enables you to check all of the check boxes in a series at the same time, rather than having to check each one individually. Take the document shown in Figure 17-4, for example.

FIGURE 17-4

A series of check boxes might benefit from an "(un)check all" check box.

When the (Un)Check All box is checked, all the check boxes will become checked. Likewise, when the (Un)Check All box is unchecked, all the check boxes will be unchecked.

This technique is accomplished using the code and document snippet shown here:

```
<html>
<head>

<script type="text/JavaScript">
function checkall() {
  var chk = form1.checks[0].checked;
  for (i = 1; i < document.form1.checks.length; i++) {
    form1.checks[i].checked = chk;
  }
}
</script>

</head>
<body>
<form name="form1" action="formhandler.cgi" method="POST">
<p><input id="allboxes" type="checkbox" name="checks"
```

```
        onClick="checkall();" />(Un)Check All</p>
<p><input type="checkbox" name="checks" />1</p>
<p><input type="checkbox" name="checks" />2</p>
<p><input type="checkbox" name="checks" />3</p>
<p><input type="checkbox" name="checks" />4</p>
<p><input type="checkbox" name="checks" />5</p>
<p><input type="checkbox" name="checks" />6</p>
</form>
</body>
</html>
```

In this case, the trigger is the `onClick` event tied to the (Un)Check All check box. When the box is clicked, it changes state — to and from being checked — and the JavaScript function `checkall()`is called. This function iterates through the check boxes in the form and sets them to the same state as the triggering check box. Hence, if the box is checked, then all the check boxes will be checked. If the box is unchecked, then all the boxes will likewise be unchecked.

Note

It is important to assign a unique name to the `form` element and its children (in this case check boxes). It enables the JavaScript code to identify and act upon that form and its elements. ■

You can use similar techniques with other form elements. For example, you could use an `onChange` event with a select box. When a new selection is made, the form could morph to suit the new selection.

Rollovers

Creating rollovers using JavaScript can be as simple or as tedious as you wish it to be. Best practice would suggest you should create rollovers, like any other JavaScript-based functionality, in a way that creates the fewest problems for the most users.

Cross-Ref

You can also create rollover effects using the CSS anchor pseudo-class `:hover`. Samples of this technique are covered in Chapter 35. ■

You can take advantage of the narrowing gap in differences among browsers by relying on the event models of the main browsers. For example, the following bit of code creates a rollover of sorts that displays a JavaScript alert box when a user mouses over a portion of text:

```
To use this rollover, <span style="color:red; cursor:hand;"
onMouseOver="alert('AMAZING!!!')"> mouse over these
words</span>.
```

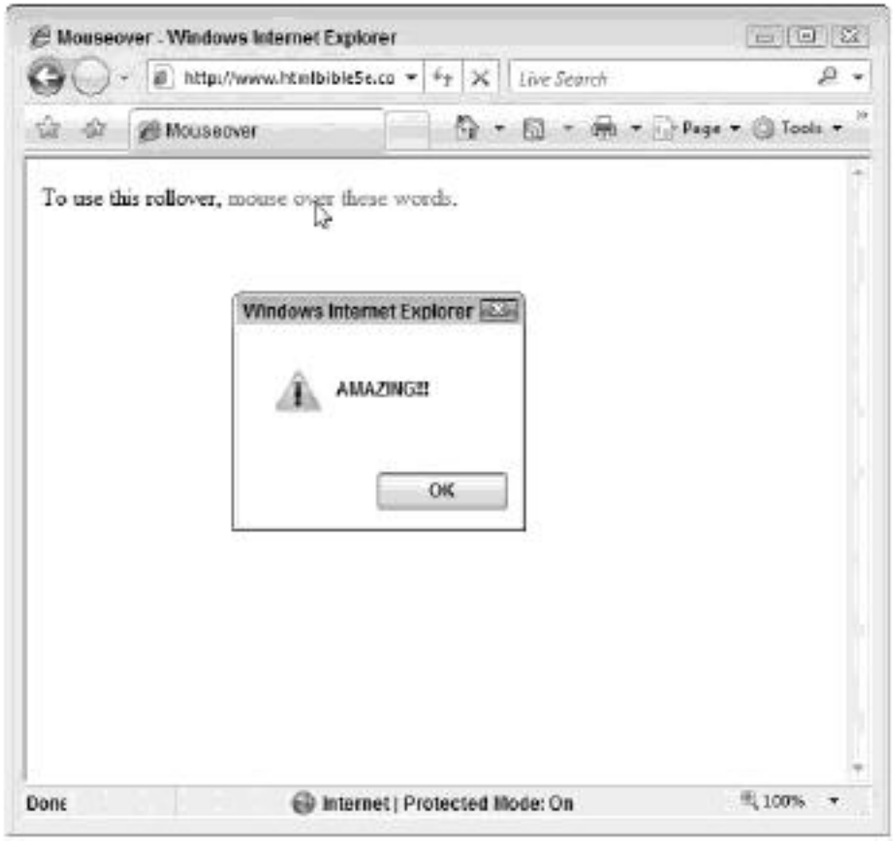
The result of this simple bit of code is shown in Figure 17-5.

Mozilla and IE allow all elements to use event handlers such as `onmouseover`. But because it's an attribute, browsers that don't support event handlers in all their elements will ignore the call

to the JavaScript because they simply ignore the attribute itself. Keep this concept in mind when you're working with DHTML. In other words, try to limit the damage. The beauty of CSS is that if you use it right, browsers that don't support CSS will simply ignore your styling. The same is true for the use of event handlers in HTML.

FIGURE 17-5

When a user mouses over a portion of text, an alert box is displayed.



Collapsible menus

Collapsible menus have become a staple in Web development, and you can generally avoid the hassle of creating your own from scratch by simply searching the Internet for something that is close to what you want; then make any adaptations necessary to reflect your own site's needs. Collapsible menus generally come in two styles:

- **Vertical menus that expand and collapse on the left side of a Web page and within a reasonably small space** — When a user clicks his or her mouse on an item, a group of one or more sub-items is displayed and, generally, remains displayed until the user clicks the main item again, which then collapses the tree.
- **Horizontal menus that live at the top of a page** — When a user rolls his or her mouse over an item, a group of one or more sub-items is displayed and, generally, disappears when the mouse loses focus on the item.

How they work

Most collapsible menus rely on either the CSS `display` property or the CSS `visibility` property. The JavaScript used to manage these menus turns the display/visibility on or off depending on where a user's mouse is, or turns the display on or off to collapse or expand a menu. The difference between the `visibility` property and the `display` property is that when you hide an element's visibility, the element still takes up visible space in the browser document. When you turn the `display` property off by giving it a `none` value (`display = "none"`), the space where the affected element lives collapses.

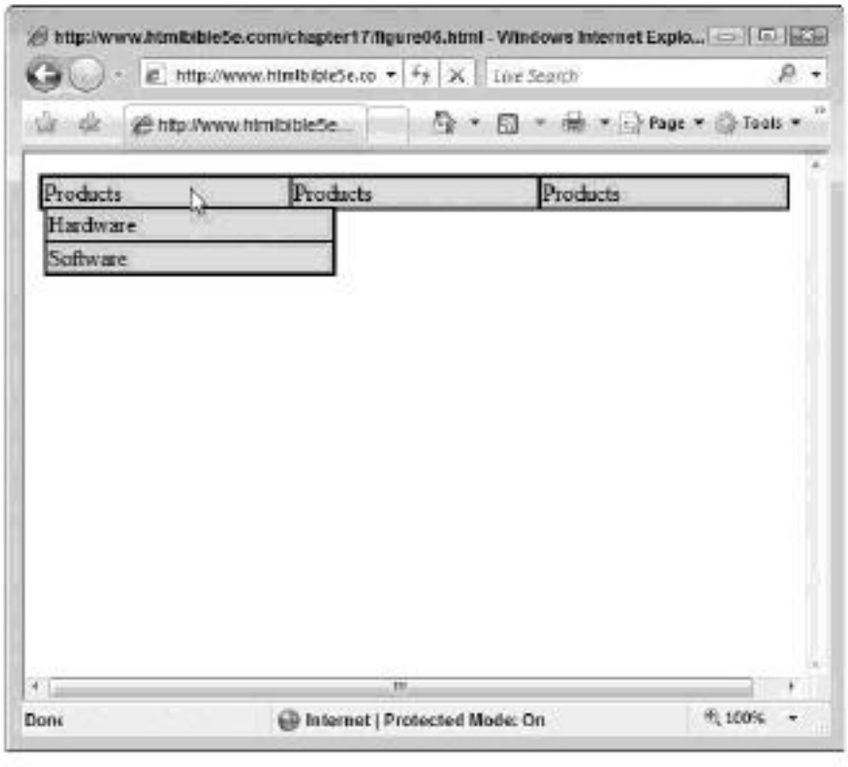
The following code shows an example of a pull-down menu, using JavaScript event triggers and a hidden table. Figure 17-6 shows the menu in action.

```
<html>
<head>
<style type="text/css">

table.topmenu { background-color: black; }

table.topmenu td  { background-color: lightblue;
                    width: 200px;  }

table.menu { background-color: black; }

table.menu td { color: black; }

</style>

<script type="text/JavaScript">

function showmenu(menu) {
  obj = document.getElementById(menu);
  obj.style.visibility = "visible";
}
function hidemenu(menu) {
  obj = document.getElementById(menu);
  obj.style.visibility = "hidden";
}

</script>
</head>
<body>
<table class="topmenu" border="0">
<tr>
  <td onMouseOver="showmenu('products');"
      onMouseOut="hidemenu('products');">Products<br />
    <table id="products" class="menu" border="0"
      style="visibility: hidden; position: absolute;">
    <tr><td>Hardware</td></tr>
    <tr><td>Software</td></tr>
    </tr>
```

```
      </table>
    </td>
    <td onMouseOver="showmenu('services');"
        onMouseOut="hidemenu('services');">Products<br />
      <table id="services" class="menu" border="0"
        style="visibility: hidden;
         position: absolute;">
      <tr><td>Documentation</td></tr>
      <tr><td>Translations</td></tr>
      </tr>
      </table>
    </td>
    <td onMouseOver="showmenu('company');"
        onMouseOut="hidemenu('company');">Products<br />
      <table id="company" class="menu" border="0"
        style="visibility: hidden; position: absolute;">
      <tr><td>About</td></tr>
      <tr><td>Contact</td></tr>
      </tr>
      </table>
    </td>
  </tr>
  </table>
  </body>
  </html>
```

FIGURE 17-6

DHTML menus can be as simple or complex as your code will allow.



The mechanics of this menu are fairly straightforward. A series of table cells is filled with the top menu item ("Products," etc.) and contains a hidden table of sub-elements (`visibility="hidden"` in the elements' `style` attribute. The `onMouseover` event is used to call the `showmenu()` JavaScript function when the user mouses over a top menu. The JavaScript function changes the embedded table's visibility to "visible," revealing the table of sub-elements. When the mouse leaves the top menu item, the `onMouseOut` event triggers the `hidemenu()` function, which changes the embedded table's visibility back to hidden, hiding the submenu.

Tip

As previously mentioned, you often don't need to write your own menu from scratch because so many developers have made them freely available. Instead, you can download someone else's menu and change the CSS and some of the other specifics, such as the location to which the links refer. ■

Summary

This chapter covered DHTML, or how you can use JavaScript along with HTML and CSS to create dynamic documents. You learned how to reference elements within the document and how to use JavaScript to interact with them.

Note that this book's primary focus is HTML and CSS, and although JavaScript and DHTML remain in the requisite coverage along with those two topics, it cannot be given the depth necessary to make one highly proficient in the subject. If JavaScript interests you, refer to a book, such as the JavaScript Bible (Wiley, 2009), that is more specific to the topic.

CHAPTER 18

The Future of HTML: HTML5

HTML has come a long way since its inception back in 1991. If you examine the specification by release version numbers alone — 2, 3.2, 4.0, and 4.01 — you might be tempted to disregard the evolutionary changes brought forth with each version. However, now that we stand on the edge of the release of HTML5, the jump in technology and intended use of the language becomes very apparent.

This chapter presents an overview of HTML5.

IN THIS CHAPTER

More Publishing and Layout Features

Accessible Multimedia

Changes: Elements and Attributes

Note

This chapter was written based on a draft specification of HTML5. Currently, this version is still several years away from release and general adoption. As such, documentation within this chapter may not exactly match the final release of HTML5. To keep tabs on the latest happenings with HTML5, visit the official W3C site: `http://dev.w3.org/html5/spec/Overview.html`. ■

More Publishing and Layout Features

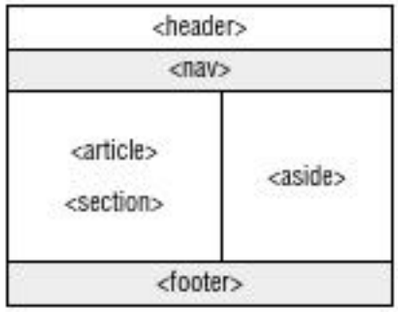
The most interesting new aspects of HTML5 are two-fold:

- Elements created for publishing purposes, not just markup
- Elements created to provide easier avenues for nontextual elements (like multimedia)

Some examples of new elements created for publishing purposes are the new `section` elements — `header`, `hgroup`, `nav`, `section`, `article`, `aside`, and `footer`. These elements are designed to free Web authors from overusing `div` elements to delimit document elements. For example, Figure 18-1 shows a suggested page layout created using the new section elements in place of `div` elements.

FIGURE 18-1

This use of the new page division tags bears a distinct resemblance to the layout of most documents on the Web.



A snippet of documents using the older `<div>` tags and newer specific section tags (e.g., `<header>`) illustrates the difference between the markups. The first listing shows the traditional, `div`-heavy method of markup, which can be hard to read and navigate. The second listing shows the same general markup but uses the new document sectioning elements:

```
<!-- Standard div layering -->
<div id="header"> ...  </div>
<div id="nav"> ... </div>
<div id="section">
    <div id="article"> ...  </div>
    <div id="article"> ...  </div>
    ...
</div>
<div id="aside"> ...  </div>
<div id="footer"> ...  </div>

<!-- New section element  layering -->
<header> ...  </header>
<nav> ... </nav>
<section>
    <article> ...  </article>
    <article> ...  </article>
    ...
</section>
<aside> ...  </aside>
<footer> ...  </footer>
```

Note

The `section` elements were created to help create documents like the one shown in Figure 18-1. As you might have guessed, blogs and news sites played a large part in this evolution. ■

Accessible Multimedia

Another big change coming with HTML5 is access to native multimedia — sound, video, and vector drawing. Although HTML5 will rely on the platform to supply the output means for the multimedia, the intent is to provide more native, and less plug-in-reliant, multimedia.

Note

Due to the large user base of certain plug-ins, such as Flash, the new multimedia features of HTML cannot hope to actually replace plug-ins. However, the new features enable other methods of creating simple multimedia options. ■

One powerful feature that has already made headlines is the canvas-drawing feature. Using a new element (`canvas`), the Web author can delimit an area within the document for drawing and use new JavaScript methods to draw within the canvas. The following code produces the canvas drawing shown in Figure 18-2:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Using the Canvas</title>
    <style type="text/css">
      canvas { border: thin solid black; }
    </style>

    <script type="text/javascript">
window.addEventListener('load', function () {

  // Get the canvas element.
  var elem = document.getElementById('myCanvas');
  if (!elem || !elem.getContext) {
    return;
  }

  // Get the canvas 2d context.
  var context = elem.getContext('2d');
  if (!context) {
    return;
  }

  // Draw a blue rectangle.
  context.fillStyle = '#00f';
  context.fillRect(0, 0, 150, 100);
  context.fillStyle = "rgba(255, 0, 0, .5)";

  // Draw a red circle with transparency
  context.beginPath();
  context.arc(200, 200, 150, 0, Math.PI*2, true);
  context.closePath();
  context.fill();
}, false);
    </script>

  </head>
  <body>
```

```
        <p><canvas id="myCanvas" width="200" height="150">Your
         browser does not have support for Canvas.</canvas></p>

    </body>
</html>
```

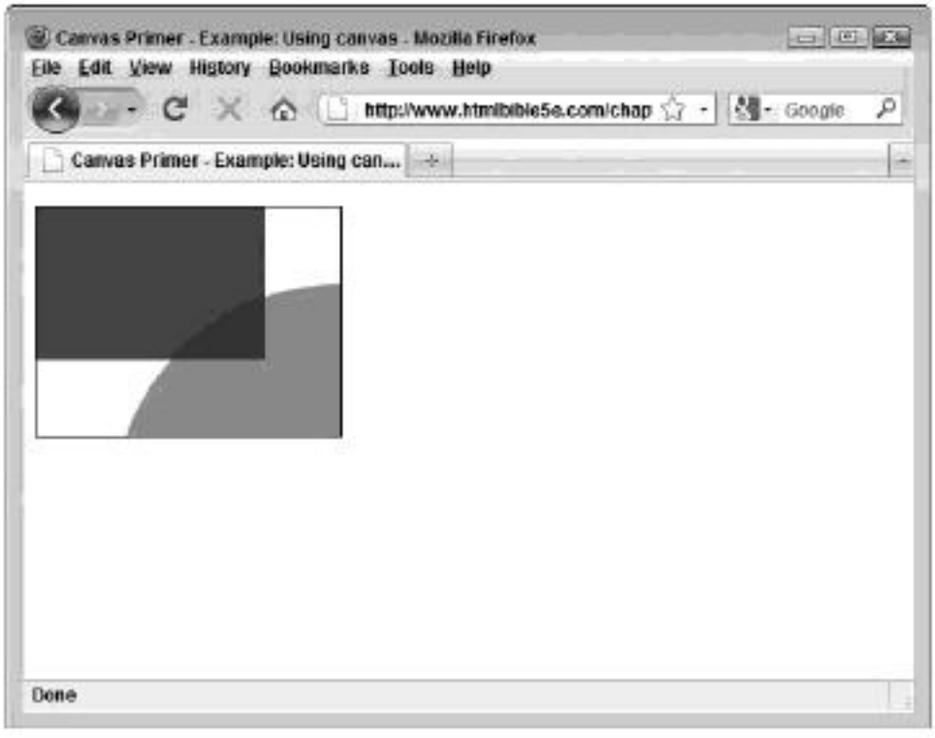
Tip

For more information on the canvas features, see the tutorial at `https://developer.mozilla.org/en/Canvas_tutorial`. ■

The `canvas` element provides a mechanism for those user agents that don't support this feature. If the agent doesn't support the canvas feature, it will display fallback content that says "Your browser does not have support for Canvas" (or whatever markup appears between the canvas tags). Note in Figure 18-2 how the canvas clips any element drawn outside of its border — the circle in this case.

FIGURE 18-2

The new canvas feature provides drawing mechanisms to HTML documents.



The `canvas` element also has a border applied. Because it is an HTML block element, it can have all the usual block element properties applied to it — positioning, decorative, and so on.

Changes: Elements and Attributes

There have been many changes to the elements and attributes that make up HTML5. The following sections provide an overview of the more noticeable changes.

Note

Throughout this book we have continually suggested XHTML formatting standards and techniques. Both will come in handy with the advent of HTML5, which has deep roots in XHTML. ■

New elements

Several new elements have been added to extend the capabilities of HTML5's markup, as shown in the following table.

Element(s)	Use
`section`	Represents a generic section of a document
`article`	Represents an independent piece of the document whole
`aside`	Represents a piece of content slightly related to the document whole
`hgroup`	Represents the header of a section
`header`	Represents a group of introductory or navigational aids
`footer`	Represents the footer of a section of the document
`nav`	Represents a section of the document intended for navigation
`dialog`	Used with `dt` and `dd` elements to mark up a conversation
`figure`	Used to provide a caption to embedded content
`video, audio`	Used to provide multimedia content
`embed`	Used to provide plug-in content
`mark`	Used to designate marked content
`progress`	Used to provide a status or progress bar
`meter`	Used to represent a measurement
`time`	Represents a date or time
`ruby, rt, and rp`	Used to provide an interface into Ruby applications
`canvas`	Used to contain rendered text or shapes
`command`	Used to reference a user-accessible command
`details`	Used to reference additional controls available to the user
`datalist`	Used to help build combo-boxes
`output`	Represents output generated by another source

New attributes by element

Several existing elements have been given additional attributes to help extend HTML5's capabilities, as shown on the following table.

Element(s)	Attribute(s)
a, area	media, ping, target
area	hreflang, rel
base	target
li	value
ol	start
meta	charset
input	autofocus (except type of hidden)
input, textarea	placeholder
input, output, select, textarea, button, and fieldset	form
input	required (except type of hidden, image, or button), autocomplete, min, max, multiple, pattern, and step
fieldset	disabled
form	novalidate
input, button	formaction, formenctype, formmethod, formnovalidate, and formtarget
menu	type, label
style	scoped
script	async
html	manifest
link	sizes
ol	reversed
iframe	seamless, sandbox

New input types (form input element)

The input element's type attribute supports several new values to aid in the input of additional values:

```
tel
search
url
email
datetime
date
```

```
month
week
time
datetime-local
number
range
color
```

These new input types provide HTML form support of new data formats without requiring additional scripting.

New global attributes

The new specification also adds more global attributes, shown in the following table, that can be applied to most elements, giving the author better control over specifying or exempting element-level features.

Attribute	Use
`contenteditable`	Marks an editable area of the document
`contextmenu`	Points to an optional context menu
`data-`	Author-defined attributes
`draggable`	Marks content as draggable (via mouse)
`hidden`	Remove element
`item, itemprop, subject`	Provides Microdata elements
`role, aria-`	Used to provide assistive technology
`spellcheck`	Indicates that the content can be spell-checked

Deprecated elements

The following elements have been deprecated either in favor of other elements or because of their frequent misuse or consistent confusion surrounding their use:

```
basefont
big
center
font
s
strike
tt
u
frame
frameset
noframes
```

acronym (use abbr instead)
applet (use object instead)
isindex
dir (use ul instead)

Deprecated attributes

The following attributes have been deprecated with HTML5 in favor of more consistent usage (such as the use of styles to produce the same effect).

Attribute	Deprecated from (Element)
rev, charset	link, a
shape, coords	a
longdesc	img, iframe
target	link
nohref	area
profile	head
version	html
Name	img (use id instead)
scheme	meta
archive, classid, codebase, codetype, declare, standby	object
valuetype, type	param
axis, abbr	td, th
Scope	td

In addition, many presentation attributes have been deprecated in favor of styles, as shown in the following table.

Attribute	Deprecated from (Element)
align	caption, iframe, img, input, object, legend, table, hr, div, h1, h2, h3, h4, h5, h6, p, col, colgroup, tbody, td, tfoot, th, thead, tr
alink, link, text, vlink	body
background	body
bgcolor	table, tr, td, th, body

Attribute	Deprecated from (Element)
border	table, object
cellpadding, cellspacing	table
char, charoff	col, colgroup, tbody, td, tfoot, th, thead, tr
clear	br
compact	dl, menu, ol, ul
frame	table
frameborder	iframe
height	td, th
hspace, vspace	img, object
marginheight, marginwidth	iframe
noshade	hr
nowrap	td, th
rules	table
scrolling	iframe
size	hr
type	li, ol, ul
valign	col, colgroup, tbody, td, tfoot, th, thead, tr
width	hr, table, td, th, col, colgroup, pre

Summary

As you can see from the information in this chapter, HTML continues to march forward toward mainstream publishing mechanisms. The rift between content (HTML) and presentation (CSS) is also becoming more pronounced, forcing Web developers to use the right tool for the right purpose. Although HTML5 is still quite a ways off, its feature set will provide some welcome changes.

Part II

HTML Tools and Variants

IN THIS PART

Chapter 19: Web Development Software

Chapter 20: Publishing Your Site

Chapter 21: An Introduction to XML

Chapter 22: Creating Mobile Documents

Chapter 23: Tidying and Validating Your Documents

Chapter 24: HTML Tips and Tricks

CHAPTER 19

Web Development Software

As you have seen throughout this book, Web development is an area rich in features. The Web has come a long way from its early beginnings as a text-only medium. As online documents get more complex, the tools to create them become more powerful. Although you can still create large, feature-rich sites with a simple text editor, using more complex and powerful tools can make the task much easier. This chapter introduces several popular tools that can help you create the best online documents possible.

IN THIS CHAPTER

Text-Oriented Editors

WYSIWYG HTML Editors

Other Tools

Note

This chapter provides several recommendations for tools you should consider for online document development. However, the recommendations are just that, recommendations. Only you can decide what tools will work best for you. Luckily, most of the tools covered in this chapter have demo versions you can download and try out for a limited time. Be sure to visit the websites referenced for each tool to get more information and perhaps even download a trial version. ■

Text-Oriented Editors

Text-oriented editors have been around since the dawn of the cathode-ray tube (CRT), the technology used in most computer display screens. However, today's editors can be quite powerful and feature-rich, doing much more than simply enabling you to create text documents. This section covers the latest in text-oriented editing.

Simple text editors

Simple text editors — such as Windows Notepad or vi on UNIX/Linux — provide an invaluable service. They enable you, without intervening features, to

easily edit text-based documents. As such, they are a logical addition to your Web development toolkit.

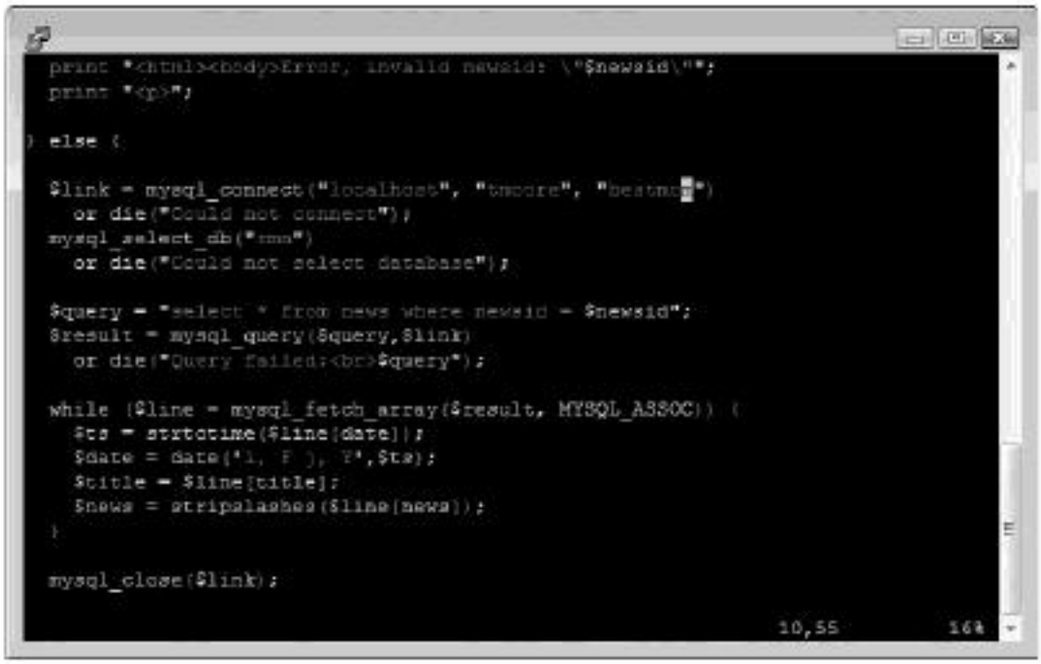
However, although you could create an entire site with one of these simple tools, there are better tools for actual creation.

Smart text editors

Smart text editors are editors that understand what you are editing and attempt to help in various ways. For example, Linux users should look into vim or Emacs and enable syntax highlighting when editing documents with embedded code (HTML, CSS, JavaScript, and so on). Figure 19-1 shows an example of a large PHP file in vim.

FIGURE 19-1

Syntax highlighting can help you avoid simple errors.



Although it may be hard to tell in the black-and-white figure, various elements have been colorized to show where they begin or end. Using methods like this, the editor keeps you abreast of what elements have been opened and which have been closed. For example, the editor may highlight quoted text in green. If most of the document turns green, it is likely that you forgot to close a quote somewhere. These editors also offer features such as auto-indenting, which can help you keep your documents structured.

Windows users have a few options for smart editors, as well. My favorite is TextPad, which uses document class templates to understand the syntax of almost any coded document.

TextPad is loaded with standard editor features. You can find TextPad on the Internet at `www.textpad.com`.

HTML-specific editors

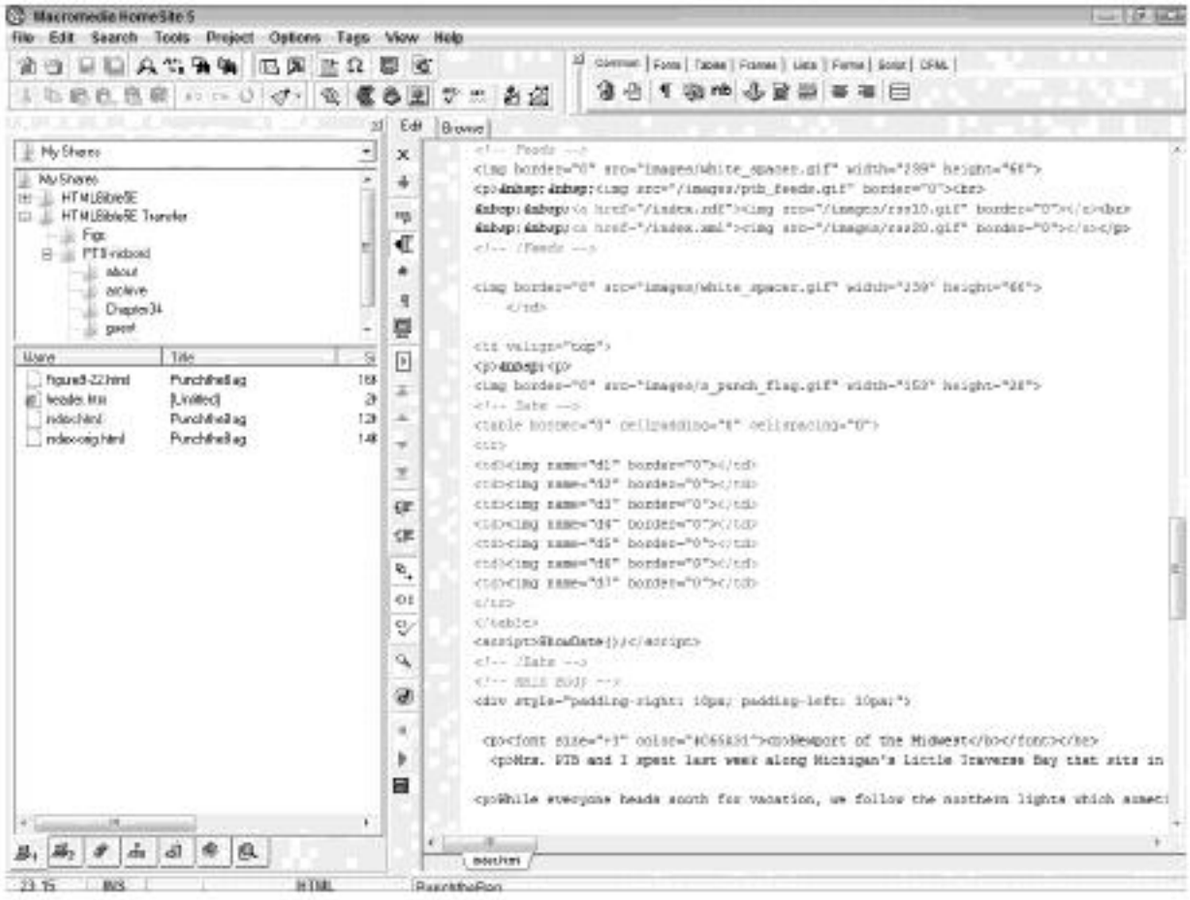
A few non-WYSIWYG editors understand HTML and provide specific features to help you code. However, HomeSite (now owned by Adobe) has always stood out from the crowd.

HomeSite provides the next level of functionality for HTML editing with special tools for entering tags and their parameters, codes for entities, macros for repeating steps, and more. Although the program is a bit dated (no support for HTML version 4.01), it is still a great choice for a full-featured HTML text editor.

Figure 19-2 shows the HomeSite main interface, and Figure 19-3 shows a tab of the dialog for creating a `<table>` tag.

FIGURE 19-2

HomeSite includes several features to make HTML editing a breeze.

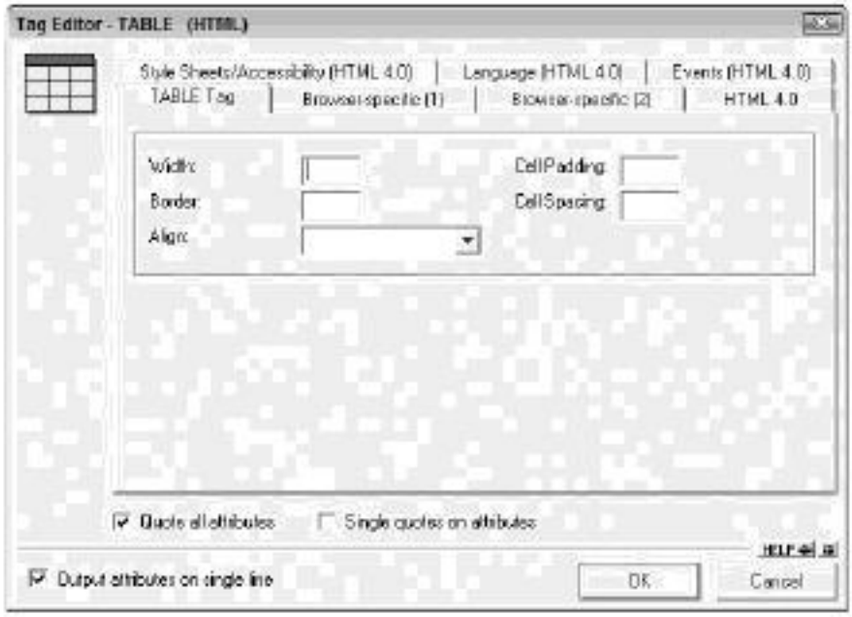


TopStyle Lite, which installs with HomeSite, provides an interface for editing and managing your external style sheets.

Visit Adobe's website for more information on HomeSite (`www.adobe.com/products/homesite/`).

FIGURE 19-3

HomeSite includes comprehensive mechanisms for building more complex tags such as tables.



Note

In 2005, Adobe Systems, Inc., acquired Macromedia lock, stock, and barrel. Most former Macromedia products are still available on the market, but under the Adobe brand name. ■

WYSIWYG HTML Editors

Just as *what you see is what you get* (WYSIWYG) editors revolutionized word processing, WYSIWYG HTML editors have revolutionized Web publishing. Using such tools, designers can design their pages visually and let the tools create the underlying HTML code. This section highlights the three most popular visual tools available for WYSIWYG editing.

NetObjects Fusion

NetObjects Fusion is another site-level design tool that offers WYSIWYG editing. The advantages of using NetObjects Fusion include easy management of entire sites, pixel-accurate designs, and a plethora of features that make publishing on the Web a breeze. Such features include the following:

- Advanced scripting support
- Automatic e-commerce catalog building