WORKING WITH HTML, CSS, AND HTTP

Working with HTML

An essential aspect of running a website is creating HTML files. Even sites that rely heavily on page technologies other than static HTML, such as script-driven pages, inevitably keep many static HTML files on the site. Although Web authors can write HTML in any text editor, many have adopted specialized HTML editor programs to handle the job.

Ideally, an HTML editor should manage the job of converting an author's writing into the structures defined by the HTML standard without requiring the author to know the details of that standard. As the job is accomplished, the editor program may carry out additional tasks, such as letting the author fine-tune the finished document's appearance or handling site management tasks such as uploading documents to a server and checking its links.

The most widely used HTML editors handle these secondary tasks with great success. Over the years, however, many designers created their actual HTML code with an emphasis on appearance rather than structure. This so-called presentational markup achieved popularity with an older generation of browsers because no widely supported alternative to controlling an HTML document's appearance was available.

Presentational HTML has drawbacks, though, and a growing number of Web authors are abandoning it in favor of HTML markup that describes a document's structure. Reasons for this choice include:

- Many aspects of presentational markup make assumptions about the user's display environment (specifically, screen resolution and window width) that are increasingly unwarranted and may lead to accessibility problems.

  A common example is pages laid out with tables of a specific width, measured in pixels. A table design that looks right *only* on a screen 800 pixels wide looks increasingly poor in screens 1,024, 1,280, or 1,600 pixels wide. Regardless of the user's screen width, the browser window displaying a page may be any fraction of that size. As screen sizes and resolutions go up, there is less likelihood that a browser will be running full-screen. Also, as described in the section on accessibility in Chapter 4, table-based layouts and other presentational effects may make a page incomprehensible to users with screen readers, and therefore inaccessible.
- The standard language for describing a document's appears, cascading stylesheets (CSS) is more powerful and flexible than presentational HTML, and good CSS design allows Web managers to apply design changes to an entire page or entire site more simply and efficiently than presentational HTML.
- Careless use of presentational HTML often introduces markup into a page that violates the HTML standard, which makes using the page impossible with any external tools that check for or require valid markup.

- Presentational HTML was created for a generation of browsers that is now so little used that authors can increasingly opt to use state-of-the-art stylesheets to give the great majority of their users richly formatted pages but still give users of older browsers fully functioning pages.
- Since 1997, the HTML standard has explicitly encouraged the cleanest possible separation between describing document structure in HTML and document appearance in CSS. Aspects of presentational markup have been officially discouraged or deprecated, since then, and are likely to disappear altogether in future versions of HTML and XHTML.

To lessen these problems, some of the most popular HTML editors suggest document appearances through stylesheets and have made generating clean, structural, valid HTML easier (even a default choice).

HTML and XHTML

Many authors are unclear about what XHTML is and how it relates to HTML. The two are similar but derived from different parent languages.

HTML was created using Standard Generalized Markup Language (SGML), which is an older standard designed with great flexibility in markup options. This standard includes features such as optional closing tags for elements such as HTML's *p* element, and more obscure syntax conventions that close multiple tags simultaneously. This flexibility was originally built into SGML to maximize its ability to interoperate with multiple proprietary publishing programs. The same flexibility, however, makes SGML a difficult format for computer programs to parse with a reliable understanding of the document structures used.

XHTML 1.0, www.w3.org/TR/xhtml1
HTML 4.01, www.w3.org/TR/html4

XHTML 1.0 is equivalent to HTML 4.01, but its parent language is XML. XML is derived from SGML but takes only a subset of SGML's features, eliminating much of the flexibility SGML provides in tagging conventions. This streamlining of features makes XML documents more consistent in their syntax, which in turn makes developing software to work with XML documents easier. Any future revisions to the HTML/XHTML languages will likely be created only as XHTML.

XHTML 1.1, which is primarily an experiment in breaking the HTML language into modular components, and an early draft of XHTML 2.0; each have only a strict mode.

In addition to differences due to their parent languages, the HTML 4.01 and XHTML 1.0 standards each provide three separate versions of their respective languages:

- A strict version, with no presentational features at all
- A transitional version with presentational features from HTML 3.2 available, but deprecated
- A frameset version that adds deprecated features for building framesets to the transitional version.

HTML editors should indicate which of these languages they are using.

Text editors and graphical editors

Historically, HTML editors have been divided into two main types: text editors and graphical editors. An advantage to using text editors is that they can be used for many different kinds of files. A Web manager can use the same application to create, for example, HTML files, cascading stylesheets (CSS) , PHP pages, and Perl scripts for logfile analysis. Text editors are not limited to the HTML elements and attributes offered by a graphical editor, which is too often a subset of what the standard provides. Most text editors do provide shortcut or macro functions to simplify inserting the most commonly used snippets of HTML markup.

An added advantage is that, with rare exceptions, some text editor is always available on a server to edit HTML. A Web manager with HTML editing experience only in Dreamweaver under Windows may be unable to make emergency updates to a page on a server without Dreamweaver available. A Web manager familiar with editing HTML with text editors can always make updates or corrections with editors available on the server.

HTML is a language of objects and modifiers. The objects (such as paragraph, table, ordered list), including their content, are elements. *The modifiers (such as class=, title=, start=) are* attributes. *In HTML, most but not all elements must have starting and ending* tags *(<p>, </p>; <table>, </table>; <ol>, </ol>). In XHTML all elements must have a starting and ending tags (including, for example, <br></br>, or alternatively <br/>.)*

The most obvious disadvantage of text editors is that they often provide too little assistance to inexperienced authors. Graphical editors typically do a better job of letting authors just start typing to ensure the result is a usable Web page. Graphical editors also constantly show the author a representation of how the page might look in a browser. Text editors do not, although they may have a command to preview a page in a browser.

Constant visual feedback may help some authors, but it may also cause them to ignore the page's structure and overlook factors that cause users with different display environments to receive a substantially different view of the page.

Graphical HTML editors emphasize an editing mode that allows authors to select HTML elements from formatting menus and displays documents as they would appear in a typical browser configuration. With graphical editors rather than text-based editors, authors are traditionally more likely to concentrate on presentational aspects of their documents.

This emphasis on presentation has been especially true of previous versions of some of the most popular graphical editors. Two widely used programs, Microsoft FrontPage and Macromedia Dreamweaver, often have been criticized for the poor quality of the HTML markup they create. Recent versions, however, provide ways for authors either to strip out presentational markup or to create clean markup by default.

Macromedia went so far as to have a team of engineers work with the Web Standards Organization to ensure Dreamweaver MX generated valid markup by default. At the same time, Microsoft worked to make sure that FrontPage 2002 could generate highly accessible pages, which reflects the need to generate clean code.

Web Standards Organization, www.webstandards.org

Making blanket statements about which single editor is best for all authors isn't possible. As with choices of server hardware and software, adopting institutional standards where they exist is more useful, as is learning to use them well rather than insisting on a particular program against the wishes of purchasing and support departments. For organizations without an editor of choice, or those in the process of deciding on one, look for, or look out for, the following features:

- Modern Web pages should be written with a clean separation between a document structure described in HTML and one or more suggested appearances described in stylesheets. An editor should either default to this approach or allow an author to select it as a configuration option. These editors also should include tools to edit cascading stylesheets.
- The same page is commonly edited at different times in different editors, some of which may understand different versions of HTML or different sets of tags. In the past many editors, on opening a document, would strip out or reformat markup they didn't recognize. Most editors now recognize that they should be able to open and save documents created in other programs without damaging the markup.
- Web authors commonly create pages on one computer, typically a desktop workstation while the website resides on a different computer—often a dedicated server. Once finished, files need to be transferred from one computer to another. Authors can transfer files with file transfer programs, including Internet Explorer, or with the command-line FTP utility available with most operating systems, but they may find the process easier if the HTML editor provides a built-in version of this function.
- Editors should offer all the HTML elements and attributes available in the version of HTML they claim to support. Even graphical editors should allow authors to edit HTML source code directly without later changing the author's markup.

FTP: File transfer protocol

Increasingly, these features are available in the most recent versions of dedicated HTML editors. They are often still lacking in general word processors or other applications with a "Save as HTML" function.

Beyond the editor: Other tools

Regardless of the editor used, authors may choose to use external utilities to polish their markup. One common tool available separately, or as a built-in feature of several editors, is HTML Tidy. Tidy is a syntax checker that catches irregularities inserted by buggy editors or mistakes in coding by hand with a text editor. HTML files that have been processed by Tidy are unlikely to retain serious markup errors and usually comply with the HTML standard.

HTML Tidy,
http:tidy.sourceforge.net

To ensure a page complies with the standard, many Web authors use a validation service. The validation process compares a document in a certain markup language, in this case HTML, against the formal definition for that language and reports every syntax error found.

Online validators are available from the World Wide Web Consortium at http://validator.w3.org and from the Web Design Group at www.htmlhelp.com/tools/validator.

Authors may be tempted to decide validation is unimportant because the page usually appears to display correctly in the author's browser. But most graphical editors create HTML with many syntax errors. A validator might report several dozen, if not several hundred, syntax errors in a page. Validation can be a frustrating process initially.

Validation is increasingly important because invalid markup cannot be guaranteed to display sensibly if the user's browser, screen resolution, or operating system differ substantially from the author's. Invalid and presentational markup both introduce problems for accessible page designs, which are required on an increasing number of websites.

Elements of modern HTML pages

Too many guides and tutorials on HTML still describe a language that predominated in the mid-1990s. Although browsers that cannot support recent developments to HTML are still in use, they are not affected by using the modern markup features described here. Because these features are supported by a large, growing number of browsers, authors should take advantage of them.

DOCTYPE declarations. One feature of a Web page that validators now require is a DOCTYPE declaration. The declaration is a formal way of announcing what version of HTML, or document type definition, a page conforms to. Identifying a page's DOCTYPE allows a validator to determine which set of rules applies to a document; in the past, including declarations was of limited benefit, since it only affected how validators looked at a page.

Two prominent browsers, Internet Explorer 6 and Mozilla (including Netscape 7) now use a page's DOCTYPE declaration, if present, to switch between a page rendering mode designed for substantially valid, structural markup and stylesheets, and one designed for backward compatibility with older authoring practices, presentational markup, and in some cases broken stylesheets.

Differences between these rendering modes can alter on-screen display of page elements; changes to margins around elements is particularly common. This behavior is written into Internet Explorer 6.0 for Windows and 5.0 for Macintosh, and in browsers based on the open-source Mozilla project, including Netscape 6 and 7. Current versions of Mozilla actually select from three rendering modes based on DOCTYPE values.

Internet Explorer's DOCTYPE-based decision making is documented at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnie60/html/cssenhancements.asp.

Mozilla's "DOCTYPE Sniffing" is documented at www.mozilla.org/docs/web-developer/quirks/doctypes.html.

Some HTML editors automatically include declarations that either are formatted incorrectly or refer to a completely wrong version of HTML. Older versions of Microsoft Frontpage often used a declaration claiming the page was written in HTML 2.0. To use Tidy on these pages, validate them, or control browser DOCTYPE behavior, edit the pages manually. Correcting the DOCTYPE declaration may be necessary.

If used, a DOCTYPE declaration must be the first line of an HTML file, or the first line output by a script after any HTTP headers. Declarations are case-sensitive, so errors in capitalization prevent validators from processing a page. Common declarations for HTML are:

HTML 4.01 transitional

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01
Transitional//EN”
”http://www.w3.org/TR/html4/loose.dtd”>
```

HTML 4.01 strict

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN”
”http://www.w3.org/TR/html4/strict.dtd”>
```

XHTML 1.0 transitional

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0
Transitional//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd”>
```

XHTML 1.0 strict

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
```

See a more complete set of possible declarations at http://validator.w3.org/sgml-lib/catalog.

Link relationships. Another feature of modern HTML that will soon become more useful is the use of link relationships. All versions of the HTML standard have defined a link element for identifying related pages and articulating what the relationship of each is. Almost every page is related to other pages. Consider this markup sample:

```
<!— This is section4.html —>
<a href=”section3.html”>Previous<a>
<a href=”section5.html”>Next</a>
```

These links refer to two related pages, and a human reader can understand what those relationships are: previous and next, but this markup has two drawbacks. First, in the entire document, only the single words *Previous* and *Next* are associated with these links, when in fact the entire section4.html document is related to these two pages. If this section of markup scrolls out of the browser window, the user has no way to navigate to the Previous and Next pages.

Second, nothing in the markup allows the browser itself to understand that these links describe these important relationships, so the browser cannot present them in a consistent way from page to page and from site to site, leaving users solely dependent on each author’s idiosyncratic style for displaying these links.

Both of these drawbacks are addressed by the link element, which applies a relationship to an entire document in a way the browser can understand, creates an interface that is always available as the user reads the page, and is consistent on all pages using link relationships.

The most widely supported link relationship is for an associated stylesheet; that link is included in a document’s head this way:

```
<link rel=”stylesheet” type=”text/css” href=”style.css”>
```

To support the possibility of other stylesheet formats, this link element must specify the type of the linked document as “text/css.” The same syntax is used for other relationships, without the MIME type; the rel attribute describes the relationship of the linked document to the current document.

Multipurpose Internet Mail Extensions (MIME) is a standard for encoding any computer data for transmission on the Internet

To express the hyperlinks above as link elements, this markup would be written into the document head:

```
<link rel=”stylesheet” type=”text/css” href=”style.css”>
<link rel=”previous” href=”section3.html”>
<link rel=”next” href=”section5.html”>
```

No firm list of allowed link relationships exists, but suggestions are written into the HTML specification. Mozilla’s site navigation bar has links for all these relationships but also supports any other relationship through a drop-down menu.

A complete set of relationships might be:

```
<link rel=”stylesheet” type=”text/css”
 href=”style.css”>
<link rel=”previous” href=”section3.html”>
<link rel=”next” href=”section5.html”>
<link rel=”first” href=”introduction.html”>
<link rel=”last” href=”conclusion.html”>
<link rel=”contents”
 href=”table-of-contents.html”>
<link rel=”bibliography” href=”references.html”>
<link rel=”author” href=”/staff/c_jones.html”>
<link rel=”up” href=”all-ebooks.html”>
<link rel=”top” href=”/welcome.html”>
<link rel=”alternate” lang=”es”
```

```
    href=”../spanish/section4.html”
    title=”In Spanish / En Espa&ntilde;ol”>
```

Historically, few browsers have ever taken advantage of relationships other than stylesheets, but two, Mozilla and Opera, have introduced versions that use HTML link elements to create navigation buttons in the browser interface. Microsoft has not addressed support for this standard in Internet Explorer.

Mozilla 1.2 and any browsers based on it pre-fetch documents with a link relationship of “next.” That is, as a user reads one document, in the background the browser downloads any document explicitly linked as the next document. Whether this feature will be entirely successful is too early to determine, but it has the potential to make websites with link relationships seem faster to users, as many documents they access will already be downloaded when they request them.

Modern forms. Starting with HTML 4.0, many features were added to the HTML language with the intent of improving accessibility. Several of these changes have the potential to make HTML forms more usable than they have usually been in the past.

The first of these is the *label* element, which associates explanatory text with a form input. Consider this section of a form:

```
<p>Inter-Library Loan Options: What format are you
borrowing?</p>
<p>
<input type=”radio” name=”format” value=”book”> Book<br>
<input type=”radio” name=”format” value=”article”>
Journal Article</p>
```

As written, graphical browsers will display a small radio button in front of the words *Book* and *Journal Article*, and will let mouse users select from those choices only by clicking on the small radio buttons. This alternative uses the label element to make this selection more usable:

```
<p>Inter-Library Loan Options: What format are you
borrowing?</p>
<p>
<input type=”radio” name=”format” value=”book”
id=”book_button”>
 <label for=”book_button”>Book</label><br>
<input type=”radio” name=”format” value=”article”
 id=”article_button”>
 <label for=”article_button”>Journal Article</label></p>
```

Although this markup doesn’t affect the appearance of the form, supporting browsers would now allow mouse users to click the words *Book* and *Journal Article* themselves to make the selection. Label phrases are often an easier target for the mouse pointer, and use of text labels for form inputs also duplicates the behavior common to dialog boxes in most applications, making Web inputs and operating system menus more consistent with each other.

Another feature that makes inputs more usable, especially in relatively complex forms, is the fieldset element. A fieldset groups a set of logically connected form inputs; graphical browsers typically put a border around them. An optional legend element gives this grouping a name, usually displayed near or on top of the border:

```
<p>Inter-Library Loan Options: What format are you
borrowing?</p>
<fieldset>
<legend>Available Formats</legend>
<input type="radio" name="format" value="book"
id="book_button">
 <label for="book_button">Book</label><br>
<input type="radio" name="format" value="article"
 id="article_button">
 <label for="article_button">Journal Article</label></p>
</fieldset>
```

Another forms-related feature that increases usability is the optgroup element. Grouping options was not added specifically for accessibility reasons, but to add organization to long drop-down lists, making them easier for all users to select from. For example, this list might grow to be unmanageable:

```
<p>Available resource guides:</p>
<option name="guide">
 <select>Art</select>
 <select>Astronomy</select>
 <select>Chemistry</select>
 <select>Geology</select>
 <select>Literature</select>
 <select>Psychology</select>
 <select>Public Health</select>
 <select>Social Work</select>
</option>
```

By contrast, in a browser that visually separates this list into groups, the list can grow longer and still remain usable:

```
<p>Available resource guides:</p>
<option name="guide">
 <optgroup label="Arts and Humanities">
 <select>Art</select>
 <select>Literature</select>
 </optgroup>
 <optgroup label="Social Sciences">
 <select>Psychology</select>
 <select>Public Health</select>
 <select>Social Work</select>
 </optgroup>
 <optgroup label="Sciences">
 <select>Astronomy</select>
 <select>Chemistry</select>
 <select>Geology</select>
 </optgroup>
</select>
```

These forms-related features are supported by current versions of Internet Explorer and Mozilla browsers, including Netscape 6 and 7. A comprehensive list of features added in HTML 4 is available in the HTML 4.01 specification.

Working with CSS

In 1996, the World Wide Web Consortium (W3C) released the first version of a language for suggesting details of a Web document's appearance. This language, cascading stylesheets (CSS), was supported incompletely or incorrectly in earlier browsers, but recent versions of all major browsers support nearly all the first versions of CSS (CSS1) and substantial parts of the second version (CSS2).

With a little care, authors can safely use CSS to provide richer formatting to their sites than was previously possible with HTML alone. They also can ensure this formatting doesn't adversely affect users of older browsers, who would see a fully functioning but plainly formatted page.

Complete or nearly complete CSS1 support is available in Internet Explorer 5.x and 6.x, Mozilla (including Netscape 6 and 7), and Opera.

By design, CSS cannot affect the display of browsers with no support for it. Several browsers have attempted to interpret CSS but they handle it with serious errors. Netscape 4.x is the most prominent of these examples.

Principles of CSS

A stylesheet is a collection of rules. Each rule consists of a selector followed by paired sets of style properties and their values. A selector, in its basic forms, can be an HTML element (all <p> elements, for example), a class of elements (<p class="database-description">), or one uniquely identified element (<p id="psycinfo">). Selectors also can be applied to any element with a certain class attribute (<p class="newsflash"> and also <h3 class="newsflash">). Other selectors are:

Grouped selectors. The same style rule can be applied simultaneously to several selectors in a comma-separated list:

```
p, td, th { font-family: Arial, sans-serif; }
```

Contextual selectors. Elements that are descendents of other elements may be selected by entering the ancestor element, a space, and the descendent element. Contextual selectors may be grouped. For example, cited titles are typically rendered with italic fonts, but if found within italicized passages they are typically rendered with roman fonts. This rule can be expressed as:

```
cite { font-style: italic; }
em cite, i cite { font-style: normal; }
```

Applying these style rules will cause the following HTML to be rendered correctly:

```
<p>The American literature discussion group will be
reading <cite>A Farewell To Arms</cite> in February.
<em>The discussion on <cite>Catch-22</cite> has been
moved to the March meeting.</em></p>
```

The American literature discussion group will be reading A Farewell To Arms *in February. The discussion on* Catch-22 *has been moved to the March meeting.*

A contextual selector title rule is available in CSS1. The CSS2 standard defines several other contexts at www.w3.org/TR/REC-CSS2/selector.html, but these contexts are not as widely supported by current browsers.

Similar contextual selectors can control the style of lists located within an item in another list (ul li ul {…}) or a paragraph within a certain class of div element (div.newsflash p {…}).

The div element marks a generic block of text, or division, within a document. It can contain other block elements such as paragraphs and lists, making it a useful tool to identify and apply styles to an entire section of a document.

Pseudo-classes and pseudo-elements. Web browsers traditionally have used typographical cues to distinguish among HTML anchor elements (<a>…</a>) that were hyperlinks from those that were named anchors in the document, and to distinguish among visited and unvisited hyperlinks. CSS addresses these differences by establishing pseudo-classes for anchors that are links and more specifically visited links, links the user's mouse pointer is hovering over, and any active links the browser is in the process of connecting to. The syntax for these pseudo-classes is:

```
a:link {…}
a:visited {…}
a:hover {…}
a:active {…}
```

Pseudo classes can be combined with normal classes and contextual selectors:

```
a:link { background: white; color: blue }
a.nav-bar:link { background: navy; color: yellow }

div.newsflash a:link, div.newsflash a:visited { font-
weight: bold; }
```

In some publishing environments, applying different styles to the first letter or first line of some blocks of text is common. CSS applies style rules to first letters and lines through two pseudo-elements, ":first-letter" and "first-line." These names are appended to normal selector names:

```
p.section-leader:first-letter { font-size: 200%; }
p.section-leader:first-line { font-variant: small-caps;
}
```

For the first-line pseudo-element, the browser determines where the first line of an element ends; the author does not need to estimate line lengths.

Properties are the typographical controls that make up the rules applied to each selector. Each property has sets of allowable values defined in the specification. Examples might include:

```
font-family: "Trebuchet MS", Arial, sans-serif;
font-size: small;
margin-left: 2em;
color: white;
background: navy;
text-align: right;
line-height: 120%;
```

A sample stylesheet might look like this:

```
/* This is a comment */
body {
 color: black;
 background: white;
 font-family: Georgia, "Times New Roman", serif;
 margin-left: 4em;
}
/* The following rule applies to all h1, h2, and h3
elements */
h1, h2, h3 {
 color: white;
 background: #009;
 font-family: "Trebuchet MS", Arial, sans-serif;
 margin-left: -2em;
}
```

These rules first establish a style for the document's <body> element. A principle of CSS is that children elements inherit many of their parents' styles. Every element in a document's body is either a child of the <body> element or of another descendent of <body>, and so inherits the style settings for <body>.

The first style rule in the previous coding includes a list of suggested fonts to use, in order of the author's preference. In this case, the rule says that the body will display with black text on a white background, using the font Georgia if it is available, and if not using Times New Roman. If Times New Roman is unavailable, the browser the system's default serif font.

The document will have a left margin 4 ems in size. (From typography, an em is a unit of length equal to the vertical distance from the top of letters such as "k" or "P" to the bottoms of letters such as "j" or "p".)

Elements have a parent-child relationship if one directly contains another. For example, in the code "<p><em>This is <strong>very</strong> important. </p>" the <em> element is the child of <p> and <p> is the parent of <em>. The <strong> element is the child of <em> and <em> is the parent of <strong>.

Predicting what fonts are available on a user's system is never completely possible. If an author wants to suggest a serif font such as Georgia or Times New Roman, the generic serif font name still selects a similar serif font if these specific fonts are unavailable. Likewise, if text is set in a sans-serif font such as Trebuchet MS or Arial, the sans-serif generic name selects a similar font if neither of those named fonts is available. If no generic font name is specified, browsers use their default font.

CSS provides many style properties; they comprise the bulk of the language. All good tutorials describe them in more detail, including www.htmlhelp.com/reference/css and www.w3schools.com/css. Another good CSS resource is the Style Sheet Reference Guide at www.webreview.com/style, which includes a master compatibility chart showing which versions of browsers support any specific CSS property.

An exception to this inheritance is established for first- through third-level headers. They will display with white text on a dark blue background, with a preferred font of Trebuchet MS if it is available, then Arial, then the system's default sans-serif font.

Colors and backgrounds. CSS can describe colors of text, background, or borders around elements. It supports several color-naming syntaxes. The simplest naming syntax, but least flexible, is to use one of 16 defined key-words: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

The other four color-naming schemes all use the computer graphics model of defining colors by their relative amounts of red, green, and blue. The first scheme, familiar to Web authors who have set colors with the HTML <font> element, is a six-digit hexadecimal number.

The first two digits specify the relative amount of red, the next two specify the amount of green, and the final two the amount of blue. Each pair of digits runs from 00 to FF, making black #000000 and white #FFFFFF; a medium blue would be #000090. A second, similar scheme is to use one hexa-decimal character per color, making black #000 and white #FFF; a medium red would be #F00.

The third color scheme describes the levels of red, blue, and green in a decimal number running from 0 to 255; this number is simply a decimal version of the six-digit hexadecimal system. The syntax for this naming scheme is rgb(x, x, x): black is rgb(0, 0, 0), white is rgb(255, 255, 255), and a dark green would be rgb(0, 40, 0). The final color scheme is similar to this but expresses each value as a percentage. Black is rgb(0%, 0%, 0%), white is rgb(100%, 100%, 100%), and a bright yellow would be rgb(100%, 100%, 0%).

Percentages. Expressing a measurement as a percentage is conceptually simple but can be made slightly more complex in CSS because different style properties are set in percentages of different source values. For example:

```
body { margin-left: 10%; }
 /* Establishes a left-hand margin that is 10% of
 the available page width */
p { line-height: 200%; }
 /* Establishes a line height that is 200% of the
 element’s font size */
ul li { font-size: 95%; }
 /* Establishes a font size that is 95% of the
 element’s parent’s font size */
```

The CSS standard or a good CSS tutorial makes clear what the percentage is being calculated against for each property.

Lengths and font sizes. Lengths also add complexity to style properties and occasionally add controversy. One problem is that the standard supports both absolute and relative measurements. Absolute measurements include inches, centimeters, millimeters, points (1/72 of an inch), and picas (1/6 of an inch).

Use of these measurements inevitably requires making assumptions about the output device that will be used to display or print a page, and Web managers cannot know that information. Absolute measurements also raise accessibility problems, because some users need to have fonts set at certain sizes just to be readable. Complicating this problem is the fact that Internet Explorer doesn’t allow users to increase or decrease font sizes that have been set with absolute values.

CSS provides two lengths that are calculated from the current font size, making them relative to the user’s font preferences. Both length specifications reflect recognized standards from the field of typography. The first is the em, a unit of length equal to the vertical distance from the top of letters such as k or P to the bottoms of letters such as j or p. The em is often mistakenly defined as a length equivalent to a font’s capital letter M. This measurement may or may not be a similar, depending on the font.

The second relative length is the x-height or ex, a measurement equal to the height of a lowercase letter x. For purposes of on-screen readability, many fonts designed for online use have greater x-heights than fonts designed for print. This design means, for example, that font sizes of Verdana and Times New Roman that have the same em size have difference ex sizes.

Applying CSS

Two common errors affect the validation process for linked stylesheets and prevent standards-compliant browsers, including Mozilla and Netscape 6 and 7, from applying them correctly. First, a standalone stylesheet must include only CSS and no HTML markup. In particular, the HTML comments often used to hide embedded stylesheets from older browsers must be eliminated. Second, Web server software must send CSS stylesheets with a MIME content type of “text/css.” Most newer servers send the correct MIME type, but some older ones may incorrectly send CSS files as “text/plain.” Apache servers typically configure MIME content types though their conf/mime.types files. IIS inherits the MIME types associated with file extensions by the server’s operating system.

Style rules can be associated with HTML markup in three locations. First, a *linked* stylesheet can stand on its own as a separate file, on either the same Web server or a different one. This external file is then referenced in the HTML page by a link element in the document's head element, explicitly describing the related document as a spreadsheet with a MIME type of "text/css":

```
<link rel="stylesheet" type="text/css" href="/styles/
main-style.css">
```

This syntax allows one stylesheet to affect appearance for all the pages on a website, or several websites, or in fact anywhere on the Web. A single revision to this stylesheet is immediately applied to all pages referencing it.

A stylesheet can also be *embedded* into an HTML document's head element. In this case, the entire stylesheet is written into the HTML document and applies only to the page containing it:

```
<style type="text/css"><!–
p.database-description {
 margin-left: 2em;
 font-size: 95%;
}
–></style>
```

Linked or embedded stylesheets can import other stylesheets in their entirety with an "@import" statement. If used, this statement must appear at the beginning of the stylesheet, before any style rules:

```
<style type="text/css"><!–
@import url(http://central.server.org/main-style.css);
p.caution {
 color: yellow;
 background: navy;
}
–></style>
```

Style rules also can be applied to individual elements with *inline* styles. These are rules applied via the style attribute allowed on most HTML elements:

```
<p style="border: thin solid red">…</p>
```

The CSS standard was written with the assumption that users would be able to create and apply their own stylesheets, in conjunction with or instead of the author's stylesheets. These stylesheets are typically selected through browser configuration settings and are not written into HTML pages.

With multiple ways to associate style rules with HTML, finding more than one stylesheet setting competing styles for a given HTML element is not uncommon. The standard determines which style to apply by establishing priorities for each rule. This priority order is the *cascade* in cascading stylesheets. When multiple rules are set, as in the CSS2 rule, the browser must:

In CSS1, author styles always took precedence over user styles.

- First, sort by weight and origin. Properties may be explicitly labeled important: User-supplied important styles take precedence, followed by author-supplied important styles, then author-supplied styles with normal weight, and then user-supplied styles with normal weight.
- Second, sort by specificity of the selector. Specificity is a calculation based on the number of elements, classes, and ID attributes addressed by a style rule. The smaller this number, the higher the priority of the style rule.
- Finally, sort by the order in which the rules are specified. When all other priorities are equal, the rule set last takes precedence. Imported rules are

considered to come before all the rules written into the stylesheet itself, and therefore have lower priority.

The Netscape 4 problem

By design, CSS is invisible to browsers that do not support it. In browsers with no CSS support at all, style properties set for a given element are ignored, and the user is shown the browser's default display for that element. Problems arise, however, with browsers that attempt to support CSS but do so with serious errors.

Netscape 6 and Netscape 7, both based on the open-source Mozilla browser, have excellent support for the CSS standard.

The many problems Netscape 4 has in supporting basic CSS rules are documented at www.webreview.com/style/css1/charts/mastergrid.shtml and http://css.nu/pointers/bugs-nn.html#NN4.

The most prominent of these browsers is Netscape 4. In user populations where this browser is still commonly used, these errors can discourage authors from using CSS at all. Note that for many CSS properties, the problem is not that Netscape 4 fails to implement support for a CSS rule, but implements it incorrectly; for example, Netscape 4 often miscalculates changes to font sizes, or applies those changes cumulatively rather than just once.

Few Web managers, though, can afford to abandon support for Netscape 4. How then can a website use modern stylesheets, understood by the great majority of browsers in use, without causing serious display problems on an aging but still relatively common browser? Netscape 4's own weak support for the CSS standard protects it from seeing style rules that would cause it to display a page incorrectly.

The most straightforward way of hiding style rules from Netscape 4 is to create one stylesheet with only the most basic rules, known to be supported in Netscape, and then import a second stylesheet with rules requiring much closer compliance with the standard. The CSS standard requires imported stylesheets be defined first in a new stylesheet, so an example might be:

```
/* Detailed CSS stylesheet imported for
standard-compliant browsers – the following
line is not understood by Netscape 4 */
@import url(full-style.css)

/* Basic colors. Include TD and TH because
Netscape 4 fails to let table cells inherit
styles from the BODY element */
body, td, th {
 background: white;
 color: black;
 font-family: Georgia, “Times New Roman”, serif;
}
```

The bulk of the style rules for this page would then actually appear in the file “full-style.css.”

A second method takes advantage of media-specific stylesheets. The CSS examples shown so far would be applied equally to on-screen displays, printed pages, and less common media such as speech synthesizers, Braille displays, and projectors. CSS2 provides mechanisms to apply style rules to all, one, or any combination of supported media, allowing, for example, details of on-screen display to be altered for printouts. Media-specific style rules are also hidden from Netscape 4, because Netscape 4 does not support rules that apply only to certain output media.

The simplest way to include media-specific rules is with the "@media" statement:

```
@media screen {
 em.highlight {
 font-style: normal;
 color: black;
 background: yellow;
 }
}
@media print {
 em.highlight {
 font-style: italic;
 color: black;
 background: white;
 }
}
```

Another use for this media-specific rules is to suppress certain sections of HTML that do not make sense in certain media:

```
@media print {
 table.navigational-tabs {
 display: none;
 }
}
```

Working with HTTP

HTTP is the Web's communications standard. The standard defines the ways that browsers and other applications make requests to servers and the ways that servers respond to those requests.

HTTP: Hypertext transfer protocol

Working with static HTML documents requires no interaction with HTTP. Authors simply put their documents in place and rely on the server's default settings to deliver the content correctly.

HTTP 1.1 standard, ftp://ftp.isi.edu/in-notes.rdc2616.txt

Web managers working with server configurations or dynamic pages, however, need to understand at least the basics of both HTTP requests and HTTP responses. With some dynamic page technologies, including CGI, scripts must provide some or all the HTTP headers the server needs to respond to the request.

A typical set of request headers may look like this set, generated by a link to http://sunsite.berkeley.edu:

```
GET / HTTP/1.0
Host: sunsite.berkeley.edu
Accept: text/html, text/plain, text/sgml, */*;q=0.01
Accept-Encoding: gzip, compress
Accept-Language: en
User-Agent: Lynx/2.8.4pre.2 libwww-FM/2.14
```

In order, these headers send the request for the server homepage (the slash) using HTTP version 1.0; the host name from which the document is re-quested, allowing the server to house multiple virtual hosts and serve differ-ent content depending on the host requested; the MIME types, document encodings or compression types, and languages the browser supports, listed in order of preference; and the User-Agent string provided by the browser to identify itself.

In response, the server sends:

```
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2002 19:07:59 GMT
Server: Apache/1.3.26 (Unix) tomcat/1.0
Connection: close
Content-Type: text/html

<HTML>…
```

By default, HTTP 1.1 connections close as soon as the document is transmitted. Browsers and servers that are fully compatible with HTTP 1.1 can establish a connection that remains open for subsequent requests, such as those needed for inline images, resulting in more efficient use of network resources.

In order, these headers show the browser that the response uses HTTP 1.1 and results in a 200 (OK) status; the date and time on the server; the server's software and version; the HTTP 1.1 connection type; and the following document's MIME type. A blank line indicates the end of the HTTP headers and the beginning of document content.

A complete description of all HTTP headers is available in the specification. Headers most relevant to website management include:

Request methods. The first header sent by the browser begins with a request method. For static pages, this method is almost always GET, which simply retrieves the document specified; scripts can use the GET method, with variables and values specified in the document's query string, or the part of a URL following a question mark. Other common methods are POST, in which form inputs are sent in a message body following the request headers, allowing for longer blocks of input data; and HEAD, which is the same as GET but returns only the HTTP headers and not the document content, allowing programs to check a document's status without having to retrieve it in its entirety. Scripts can alter their behavior based on the request method used. For example, the same script can print out an HTML form when requested with GET, and then process the results of that form when requested with POST.

A source of confusion about content types is the fact that Internet Explorer, in violation of the HTTP standard, second-guesses many common content types. For example, a URI with a type of "text/plain" is required to be treated as plain text, but IE examines its contents, and, if it decides the file appears to consist of HTML, renders it as an HTML document.

MIME types. HTTP does not use the concepts of files and file types, but rather of streams of data identified with a certain Multipurpose Internet mail extensions (MIME) content type. MIME is a standard for encoding any computer data for transmission on the Internet; encoded data are always identified by their major type (common types are text, image, or application) and specific subtype (.html, .jpeg, or .pdf). Types and subtypes are separated by a slash, making the header for HTML data "Content-type: text/html."

When addressing static files, a server usually determines content types based on file names or extensions. An example is sending all files ending in .jpg with a content type of "image/jpeg," But dynamic page technologies, which can be used to deliver any type of content, can set explicit content types. CGI scripts must always include a content type, but PHP, for example, must include it only if it's something other than "text/html."

Many Web managers have been taught that browsers always offer to save documents to a file rather than display them if the documents are sent with a content type of "application/octet-stream." This technique is often used by library databases to download search results. But defining application/octet-stream as the content type that make a browser save the file is inexact.

The MIME standard defines the stream as binary data of unknown type, and the HTTP standard suggests but does not require that application/octet-stream result in an option to save it to a file. This content type is one that

Internet Explorer examines to determine if it is a known format. Internet Explorer therefore displays plain text or HTML delivered as application/octet-stream instead of automatically offering to save it.

Instead of relying solely on this content type to open a "Save As…" prompt, Web managers should use the prompt with the Content-Disposition header, which is HTTP's mechanism for telling the browser to do something other than display document content. These headers tell a compliant browser that the following document content is plain text, but give the user a way to save the document, offering "search-results.txt" as a default filename, instead of displaying it:

```
Content-Type: text/plain
Content-Disposition: attachment; filename=
"search-results.txt"
```

Accept headers. As shown in the sample request headers above, browsers send several prioritized lists of preferences. The Accept header lists MIME types the browser is prepared to accept. A dynamic page with copies of an image in several different formats could use this header to determine whether to send a PNG or GIF version, for example. Different content types can be prioritized with a qvalue attribute assigning a preference level between a maximum default value of 1 and a minimum of 0.

Portable network graphics (PNG) is a file format supported by all major graphical browsers that is patent- and royalty-free and supports up to 16 million colors.

Graphic interchange format (GIF) is limited to 256 colors. Its compression algorithm is a patent owned by Unisys, which charges software developers royalties for adding functions such as "Save as GIF."

Similarly, the Accept-Language header expresses a user's language preferences using a standard list of two-letter codes for languages. Servers or scripts that respond to this header can allow multilingual sites to make informed guesses about which document versions to display by default. Where English and Spanish versions of a document are both available, a dynamic page could view this Accept-Language header and decide to send the English version:

```
Accept-Language: en; q=0.5 es
```

The following header, on other the hand, establishes a first language preference of Portuguese, followed by Spanish, and then English. The same dynamic page could check if a Portuguese version was available; if not, it would send the Spanish version:

```
Accept-Language: pt; q=0.8 es; q=0.5 en
```

In addition to dynamic page technologies making decisions based on Accept headers, the Apache server itself can be configured to identify and serve alternate versions of documents based on these headers. The advantage to using the server itself to react to accept headers is that authors can provide multilingual or multiformat options without needing to use or know any dynamic page technologies.

Content negotiation is described at http://httpd.apache.org/docs-2.0/content-negotiation.html.

Cookies. A cookie is a short string of text, in the form "name=value." Cookies have some features that are unique in HTTP. A basic problem in creating information services using HTTP is that HTTP is stateless: once it finishes sending a document and closes a connection, it has no mechanism to associate the same user's next request with the previous one. Statelessness prevents HTTP, by itself, from managing user sessions in any service that requires knowledge of the history of the user's requests.

This issue can be addressed in several ways. Some services use dynamic pages that maintain information on the server about what is happening, for example, in each search session. These pages associate an ID number with each session and use that ID number in all URLs. Other simpler services write only enough information into each URL to be able to reconstruct the user's location on each request.

Cookies also can keep each user associated with server sessions, among other tasks. Although some browsers allow JavaScript code to set cookie values, most services set them with HTTP headers.

What makes the Cookie header unique is that the server may send it to the browser along with directives specifying what pages, or domain, it should be sent back to, and an expiration date until which the browser should send back the cookie. That header, sent as part of a server's response, would be:

```
Set-Cookie: UserID=12345; domain="/search";
expires= Fri, 01-Jan-2010 00:00:00 GMT
```

A browser that accepts this cookie stores its value and continues to send it back to the specified pages until its expiration date, which may be years in the future. All subsequent requests to pages on the same server, in the "/ search" directory, will include this header:

```
Cookie: UserID=12345
```

Web managers have often set unique cookie values for any browser that requests documents without already having that cookie; that browser's future use of the site can be tracked by logging all requests accompanied by that cookie.

Although cookies can maintain session information and provide long-term tracking of usage patterns, growing numbers of users perceive overuse of cookies as an invasion of their privacy. Both newer browsers and third-party software give users flexibility in controlling what cookies their browsers store, and for how long. Web managers cannot fairly assume that cookies they set will be accepted, and in designing services must make allowances for users who do not return cookies set by the server.

Status codes and link checking. All responses to HTTP requests begin with a line that includes a three-digit code indicating the status of the response. These codes indicate whether the requested document was found and can be sent, or if it was moved to a new location, or if some error prevented the server from sending it.

Understanding status codes assists Web managers in several ways. Setting up server configurations and dynamic pages often requires setting certain URIs to send specific status codes, especially when pages redirect users to other locations. Status codes also are included in server log files, making possible the finding of all requests made to the server for pages that do not exist. Status codes also report on the availability of pages linked to on other servers.

Uniform Resource Identifier (URI) is the generic term for names that identify Web resources. A URL (Uniform Resource Locator) is a specific kind of URI that includes a protocol

Most websites maintain links to external resources, and the Web's nature is that these resources may move or go out of existence. Many Web managers make a regular task of ensuring all their links still point to the correct resource.

To a large extent, this check can be automated by programs that view a local page, or crawl through all the pages on the local site, compile lists of external links, and then check the status of all the pages linked to by the local pages. Although these programs may report additional data, the primary piece of information they collect from each page is its HTTP status code.

Status codes are divided into groups, identifiable by their first digit. 1xx codes accompany informational message; 2xx codes indicate a successful status; 3xx codes indicate that a redirection to another location; 4xx codes indicate an error originated on the browser's end of the request; and 5xx errors indicate a problem on the server itself. Full details are found in the specification, but several are important in the context of link checking.

The 200 (“OK”) code indicates the requested document was found and can be sent. The 301 (Moved Permanently) and 302 (Found) headers both indicate that requested document has moved, and both are accompanied by a Location header giving the new URL. The 302 status indicates a temporary redirection, but as the name implies the 301 redirection indicates the requested document has been permanently relocated.

Link checkers include in their reports both the current URL and its new location, so the document making the link can be updated. The 301 status also causes most Web crawlers and indexes to drop the old location and index the new one instead, and some browsers (notably Internet Explorer) automatically update user bookmarks when a requested document returns a 301 status.

Link checkers also report any 4xx codes received, especially the 404 (Not Found) code. This common code indicates the server currently has no document matching the one requested, and usually indicates a page, or an entire site, has moved without establishing 301 redirects to the new location.

Because link checkers and other automated HTTP applications rely so heavily on accurate status codes, Web managers must not move their own pages or sites using only warning messages intended for human readers. A page that returns a 200 status, but reads “We have moved…” cannot be relied on to update links on other sites.

Many Web managers coordinate moves in several phases, with a first phase using “We have moved” messages for users, and a later phase relying on 301 redirects for automated scripts. This later phase may need to last for a year or more to ensure that links on other sites do not take a significant number of users to the dead end of a 404 message.

Ideally, Web managers should seldom or never change the URIs for their pages. Both Apache and IIS allow managers to reorganize or move directory structures on the server and still associate their original URIs with the new locations. Breaking users’ links and bookmarks simply to reflect internal organizational changes in a new URI is never responsible.

Cache control. The current HTTP version, HTTP 1.1, devotes much attention to the ways documents can be requested and delivered through a string of proxy or cache servers. For example, a browser can go through these steps:

1. Check its own cache of the user’s recently accessed documents.
2. Access the Internet through an office firewall that caches documents received by everyone in the office.
3. Connect outward again through an Internet service provider with another cache keeping recent copies of documents requested by all subscribers.
4. Contact the server where the document originates.

Because communication with any of these caches is typically faster than communication with the originating server, caches often provide a performance benefit to users: every page retrieved from a cache appears to the user to be delivered faster than pages from the server.

Having multiple copies of a document available, however, with the potential for each copy to have been retrieved at different times, leads to problems when the original document's content changes regularly. Discrepancies among these versions can be especially problematic for dynamic pages, where the same URL may provide different content almost every time it is viewed. Fortunately, these pages have the greatest ability to set their own HTTP header to control cache behavior.

HTTP 1.1's Cache-Control header provides flexible control over cache behavior. This header can specify that content may be cached, may not be cached, or may be cached only in individual user's caches but not in shared caches (or vice versa), or cached only until a certain time. The simplest form of this header signals all caches handling a response that they are not allowed to retain a copy.

```
Cache-Control: no-cache
```

HTTP 1.0 provided only a single option for cache control: "Pragma: no-cache". Its meaning is identical to "Cache-Control: no-cache." To control the small number of HTTP 1.0 browsers and caches still in use, Web managers may choose to send both of these headers with pages that should not be cached.

If a page's output is tailored for individual users but does not change frequently, Web managers may choose to let individuals store copies in their personal caches—usually a browser component—but prevent copies from being stored in multiuser caches. HTTP refers to these two kinds of caches as private and public. Pages available for private caches only carry this header:

```
Cache-Control: private
```

When a page's output changes frequently but not constantly, Web managers can choose to let caches store a copy for a limited time. The maximum age of copies to be kept in caches can be set, in seconds. This header indicates that the page being sent should be cached for no more than one hour (3,600 seconds):

```
Cache-Control: max-age=3600
```

Pages that should be cached only until a certain date should be sent with an Expires header, which specifies the date beyond which the cached copy should be replaced with a fresh copy from the server:

```
Expires: Fri 28 Nov 2003 16:00:00 GMT
```