CHAPTER 1

JavaScript's Role in the World Wide Web

In This Chapter

How JavaScript blends with other Web authoring technologies

The history of JavaScript

What kinds of jobs you should and should not entrust to JavaScript

For the many individuals responsible today for content on the World Wide Web, it wasn't long ago that terms such as *HTML* (Hypertext Markup Language) and *URL* (Universal Resource Locator) seemed like words from a foreign language. Growth in activity on the Web — producing content and surfing it — has been nothing short of phenomenal. Some may even call the hyperactivity "scary." I could quote the estimates of the number of Web sites available on the Internet, but that count would be woefully outdated before these words ever reached a printing press.

Developers of Web software technologies are now in a desperate race to catch up with the enthusiasm that people have for the Internet — and for the Web in particular. Web site authors are constantly seeking tools that will make their sites engaging (if not "cool") with the least amount of effort. This is particularly true when the task is in the hands of people more comfortable with writing, graphic design, and page layout than with hard-core programming. Not every Webmaster has legions of experienced programmers on hand to whip up some special, custom enhancement for the site. Nor does every Web author have control over the Web server that physically houses the collection of HTML and graphics files. JavaScript brings programming power within reach of anyone familiar with HTML, even when the server is a black box at the other end of a telephone line.

Competition on the Web

Web page publishers revel in logging as many visits to their sites as possible. Regardless of the questionable accuracy of Web page *hit* counts, a site consistently logging 10,000 dubious hits per week is clearly far more popular than one with 1,000 dubious hits per week. Even if the precise number is unknown, relative popularity is a valuable measure.

Encouraging people to visit a site frequently is the Holy Grail of Web publishing. Competition for viewers is enormous. Not only is the Web like a million-channel television, but the Web competes for viewers' attention with all kinds of computer-generated information. That includes anything that appears on-screen as interactive multimedia.

Users of entertainment programs, multimedia encyclopedias, and other colorful, engaging, and mouse-finger-numbing actions are accustomed to high-quality presentations. Frequently, these programs sport first-rate graphics, animation, live-action video, and synchronized sound. In contrast, the lowest-common-denominator Web page has little in the way of razzle-dazzle. Even with the help of recent advances in Dynamic HTML, the layout of pictures and text is highly constrained compared to the kinds of desktop publishing documents you see all the time. Regardless of the quality of its content, a vanilla HTML document is flat. At best, interaction is limited to whatever navigation the author offers in the way of hypertext links or forms whose filled-in content magically disappears into the Web site's server.

Stretching the Standards

As an outgrowth of *SGML* (Standard Generalized Markup Language), HTML is generally viewed as nothing more than a document formatting, or *tagging*, language. The tags (inside <> delimiter characters) instruct a viewer program (the *browser* or the *client*) how to display chunks of text or images.

Relegating HTML to the category of tagging language does disservice not only to the effort that goes into fashioning a first-rate Web page, but also to the way users interact with the pages. To my way of thinking, any collection of commands and other syntax that directs the way users interact with digital information is *programming*. With HTML, a Web page author controls the user experience with the content, just as the engineers who program Microsoft Excel craft the way users interact with spreadsheet content and functions.

Unfortunately, the HTML standards agreed to by industry groups leave much to be desired in the way HTML programmers can customize the level of interactivity between document and user. Software companies that develop browsers feel the urgency to move the facilities of HTML forward to meet the demands of Web authors who want more control over the display of various kinds of information. Browser companies are engaged in a constant game of leapfrog as they race ahead of still-emerging standards. Even before an HTML standard version is released to the public, aggressive and creative companies such as Netscape and Microsoft implement their own extensions to stretch the state-of-the-art.

CGI Scripting

One way to enhance the interaction between user and content is to have the page communicate with the Web server that houses the Web pages. Popular Web search sites, such as Yahoo!, Digital's Alta Vista, and Lycos, let users type search criteria and click a button or two to specify the way the search engine should treat the query. When you click the Submit or Search buttons, your browser sends your entries from the form to the server. On the server, a program known as a CGI

(Common Gateway Interface) script formats the data you've entered and sends it to a database or other program running on the server. The CGI script then sends the results to your browser, sometimes in the form of a new page or as information occupying other fields in the form.

Writing customized CGI scripts typically requires considerable programming skill. It definitely requires the Web page author to be in control of the server, including whatever *back-end* programs, such as databases, are needed to supply results or to massage the information coming from the user. Even with the new, server-based, Web site design tools available, CGI scripting often is not a task that a content-oriented HTML author can do without handing it off to a more experienced programmer.

As interesting and useful as CGI scripting is, it burdens the server with the job of processing queries. A busy server may be processing hundreds of CGI scripts at a time, while the client computers — the personal computers running the browsers — are sitting idle as the browser's logo icon dances its little animation. This wastes desktop processing horsepower. That's why some people regard browsing a basic Web page as little more than using a dumb terminal to access some server content.

Of Helpers, Plug-ins, and Applets

Not that long ago, browsers relied on helper applications to make up for their internal deficiencies. For example, browsers typically don't know how to deal with audio that comes in from a Web site. Instead, your browser knows that when it encounters an audio file, it must launch a separate helper program (if available on your hard disk) to do the job of making your particular flavor of computer (Windows 3.*x*, Windows 95, UNIX, or MacOS) convert the digitized sound to audio you can hear through speakers.

A current trend in browsers is to bring as much functionality as possible into the browser itself, so you don't have to run another program to play a movie or audio clip. This integration also helps in giving the overall presentation a cleaner appearance to the user. Instead of having to play in a separate window atop the browser's document window, a movie can appear in the page, perhaps accompanied by label text or a caption.

Plug-ins

First available in Navigator 2, software plug-ins for browsers enable developers to incorporate a variety of capabilities into the browser without having to modify the browser. Unlike a helper application, a plug-in is usually less demanding of client resources and enables external content to be blended into the document seamlessly.

The most common plug-ins are those that facilitate the playback of audio and video from the server. Audio may include music tracks that play in the background while visiting a page or live audio similar to a radio station. Video and animation can operate in a space on the page when played through a plug-in that knows how to process such data.

Java applets

When the interaction required between user and Web page exceeds the capabilities of HTML, experienced programmers may prefer to "roll their own" programs to handle the special needs not available in existing plug-ins. Filling this need is the Java programming language. Developed at Sun Microsystems, the language enables programmers to write small applications (*applets*) that download to the browser (as separate files, such as image files). An applet runs as the user needs it and then is automatically discarded (from memory) when the user moves elsewhere in the Web.

Animation, including animated text whose content can change over time, is a popular application of the Java applet in an HTML page. Because applets can also communicate with the Internet as they run, they are also used for real-time, data streaming applications, displaying up-to-the-minute news, stock market, and sports data as it comes across the wires. All of this activity can be surrounded by standard HTML content as the Web page designer sees fit.

To play a Java applet, a browser company must have licensed the technology from Sun and built it into its browser. Netscape was the first third-party browser supplier to license and produce a browser capable of running Java applets (Navigator 2 under Windows 95 and UNIX). Today, both Netscape Navigator and Microsoft Internet Explorer (IE) can load and run Java applets on almost every operating system platform supported by the browser. The perceived popularity of Java is so strong today that the capability of running Java-based programs is being built or planned not only for browsers (such as Internet Explorer), but for entire operating systems, such as the MacOS and Windows 98/NT.

JavaScript: A Language for All

The Java language is derived from C and C++, but it is a distinct language. Its main audience is the experienced programmer. That leaves out many Web page authors. I was dismayed at this situation when I first read about Java's specifications. I would have preferred a language that casual programmers and scripters who use authoring tools such as ToolBook, HyperCard, and even Visual Basic could adopt quickly. As these accessible development platforms have shown, nonprofessional authors can dream up many creative applications, often for very specific tasks that no programmer would have the inclination to work on. Personal needs often drive development in the classroom, office, den, or garage. But Java was not going to be that kind of inclusive language.

My spirits lifted several months later, in November 1995, when I heard of a scripting language project brewing at Netscape. Initially born under the name LiveScript, this language was developed in parallel with Netscape's Web server software. The language was to serve two purposes with the same syntax. One purpose was as a scripting language that Web server administrators could use to manage the server and connect its pages to other services, such as back-end databases and search engines for users looking up information. Extending the "Live" brand name further, Netscape assigned the name LiveWire to the database connectivity usage of JavaScript on the server.

On the client side — in HTML documents — these scripts could be used to enhance Web pages in a number of ways. For example, an author could use LiveScript to make sure that the information a user entered into a form would be of the proper type. Instead of forcing the server or database to do the data validation (requiring data exchanges between the client browser and the server), the user's computer handles all the calculation work — putting some of that otherwise wasted horsepower to work. In essence, LiveScript could provide HTML-level interaction for the user.

As the intensity of industry interest in Java grew, Netscape saw another opportunity for LiveScript: as a way for HTML documents (and their users) to communicate with Java applets. For example, a user might make some preference selections from checkboxes and pop-up selection lists located at the top of a Web page. Scrolling down to the next screenful, the user sees text in the Java applet scrolling banner on the page that is customized to the settings made above. In this case, the LiveScript script sends the text that is to appear in the scrolling banner to the applet (and perhaps a new color to use for the banner's background and text). While this is happening, the server doesn't have to worry a bit about it, and the user hasn't had to wait for communication between the browser and the server. As great an idea as this was initially, this connectivity feature didn't make it into Navigator 2 when JavaScript first became available.

LiveScript becomes JavaScript

In early December 1995, Netscape and Sun jointly announced that the scripting language would thereafter be known as JavaScript. Though Netscape had several good marketing reasons for adopting this name, the changeover may have contributed more confusion to both the Java and HTML scripting worlds than anyone had expected.

Before the announcement, the language was already related to Java in some ways. Many of the basic syntax elements of the language were reminiscent of the C and C++ style of Java. For client-side scripting, the language was intended for very different purposes than Java — essentially, to function as a programming language integrated into HTML documents, rather than as a language for writing applets that occupy a fixed rectangular area on the page (and that are oblivious to whatever else may be on the page). Instead of Java's full-blown programming language vocabulary (and conceptually difficult object-oriented approach), JavaScript had a small vocabulary and more easily digestible programming model.

The true difficulty, it turned out, was making the distinction between Java and JavaScript clear to the world. Many computer journalists made major blunders when they said or implied that JavaScript was a simpler way of building Java applets. To this day, many programmers believe JavaScript to be synonymous with the Java language: They post Java queries to JavaScript-specific Internet newsgroups and mailing lists.

The fact remains today that Java and JavaScript are more different than they are similar. Java support in a browser or operating system does not automatically imply JavaScript support. The two languages require entirely different interpreter engines to execute their lines of code. Whereas JavaScript support shipped in every platform-specific version of Navigator 2 in February 1996, Java was not available for Windows 3.1 users until late in the life of Navigator 3. (Many

squirrelly technical issues make it difficult for this modern language to work in an "ancient" MS-DOS operating system.)

Coming together

Now that I've made such a point of distinguishing JavaScript from Java, I'm here to tell you that the two languages are more complementary than ever since the release of Navigator 3. Detailed at length later in the book, a Netscape technology called *LiveConnect* makes it possible for JavaScript to communicate with Java applets (as well as perform some Java functionality directly). JavaScript writers can treat prewritten Java applets as ready-to-run programs that require very little in the way of coding to modify and control.

By adding scriptability to Java applets (when applets have been written to be scriptable), LiveConnect significantly increases the powers of HTML authors and scripters, without requiring them to trudge up the steep learning curve of the full Java language. Java applets can do their animation and highly powered algorithmic tasks, whereas we scripters control how the applets behave in response to user interaction with HTML elements, such as buttons and select lists. It truly is a win-win-win scenario for scripters, Java applet programmers, and the users of our pages.

The Microsoft world

Anyone who follows the Web software industry knows about the race for market superiority involving players such as Netscape, Microsoft, and Sun Microsystems. Each company has contributed to the user enjoyment of the Internet and will continue to do so in the "Web Weeks" ahead (a measure of time coined by Eric Schmidt while he was at Sun).

As a result of Microsoft being a comparative latecomer to the Internet party, it has embarked on a philosophy that it calls "embrace and extend." This means that even if Microsoft prefers its own technologies to market-leading products developed outside of Microsoft, the company will find ways to enable those outside technologies to work within its products. As solid demonstrations of that philosophy, Microsoft implemented both Java and JavaScript for release 3 of its Internet Explorer browser. The company has greatly enhanced the JavaScript offering in Internet Explorer 4.

In keeping with the competitive nature of the Web browser market, Netscape and Microsoft continue to attract developers to their camps with unique extensions in each new version. If you are developing pages for an audience that uses both browser brands, this creates challenges. I address these issues in the next chapter.

JavaScript: The Right Tool for the Right Job

Knowing how to match an authoring tool to a solution-building task is an important part of being a well-rounded Web page author. A Web page designer who ignores JavaScript is akin to a plumber who bruises his knuckles by using pliers instead of the wrench at the bottom of the toolbox. By the same token, JavaScript won't fulfill every dream.

The more you understand about JavaScript's intentions and limitations, the more likely you will be to turn to it immediately when it is the proper tool. In particular, look to JavaScript for the following kinds of solutions:

- ✦ You want your Web page to respond or react directly to user interaction with form elements (input fields, text areas, buttons, radio buttons, checkboxes, selection lists) and hypertext links — a class of application I call the *serverless CGI*.
- ✦ You want to distribute small collections of database-like information and provide a friendly interface to that data.
- ✦ You need to control multiple-frame navigation, plug-ins, or Java applets based on user choices in the HTML document.
- ✦ You want data preprocessed on the client before submission to a server.

At the same time, understanding what JavaScript is not capable of doing is vital. Scripters waste many hours looking for ways of carrying out tasks for which JavaScript was not designed. Most of the limitations are designed to protect visitors from invasions of privacy or unauthorized access to their desktop computers. Therefore, unless a visitor is using a modern browser and explicitly gives you permission to access protected parts of his or her computer, JavaScript cannot surreptitiously perform any of the following actions:

- ✦ Setting or retrieving the browser's preferences settings, main window appearance features, action buttons, and printing
- ✦ Launching an application on the client computer
- ✦ Reading or writing files or directories on the client computer
- ✦ Extracting the text content of HTML pages or their files from the server
- ✦ Writing files to the server
- ✦ Reading a server directory
- ✦ Capturing live data streams from the server
- ✦ Sending secret e-mails from Web site visitors to you

Beyond the security issues, should you turn to JavaScript and find that it doesn't have the capabilities you need, speak up! Let the JavaScript development team at Netscape know what you'd like it to do in the future. JavaScript will certainly evolve and grow as scripters stretch its powers.

✦ ✦ ✦