

Chapter 1

JavaScript's Role in the World Wide Web and Beyond

Many of the technologies that make the World Wide Web possible have far exceeded their original goals. Envisioned at the outset as a medium for publishing static text and image content across a network, the Web is forever being probed, pushed, and pulled by content authors. By taking for granted so much of the “dirty work” of conveying the bits between server and client computers, content developers and programmers dream of exploiting that connection to generate new user experiences and practical applications. It's not uncommon for a developer community to take ownership of a technology and mold it to do new and exciting things. But with so many Web technologies — especially browser programming with JavaScript — being within reach of everyday folks, we have witnessed an unprecedented explosion in turning the World Wide Web from a bland publishing medium into a highly interactive, operating system-agnostic authoring platform.

The JavaScript language, working in tandem with related browser features, is a Web-enhancing technology. When employed on the client computer, the language can help turn a static page of content into an engaging, interactive, and intelligent experience. Applications can be as subtle as welcoming a site's visitor with the greeting “Good morning!” when it is morning in the client computer's time zone — even though it is dinnertime where the server is located. Or applications can be much more obvious, such as delivering the content of a slide show in a one-page download while JavaScript controls the sequence of hiding, showing, and “flying slide” transitions while navigating through the presentation.

Of course, JavaScript is not the only technology that can give life to drab Web content. Therefore, it is important to understand where JavaScript fits within the array of standards, tools, and other technologies at your disposal. The alternative technologies described in this chapter are HTML, Cascading Style Sheets (CSS), server programs, and plug-ins. In most cases, JavaScript can work side by side with these other technologies, even though the hype around some make them sound like one-stop shopping places for all your interactive needs. That's rarely the case. Finally, you learn about the origins of JavaScript and what role it plays in today's advanced Web browsers.

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

Competing for Web Traffic

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. Another useful number is how many links from outside pages lead to a site. A popular site will have many other sites pointing to it—a key to earning high visibility in Web searches.

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 50 million–channel television, but also, the Web competes for viewers' attention with all kinds of computer-generated information. That includes anything that appears onscreen 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. By contrast, the lowest-common-denominator Web page has little in the way of razzle-dazzle. Even with the help of Dynamic HTML and stylesheets, the layout of pictures and text is highly constrained compared with the kinds of desktop publishing documents you see all the time. Regardless of the quality of its content, an unscripted, 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.

Other Web Technologies

With so many ways to spice up Web sites and pages, you can count on competitors for your site's visitors to do their darnedest to make their sites more engaging than yours. Unless you are the sole purveyor of information that is in high demand, you continually must devise ways to keep your visitors coming back and entice new ones. If you design for an intranet, your competition is the drive for improved productivity by colleagues who use the internal Web sites for getting their jobs done.

These are all excellent reasons why you should care about using one or more Web technologies to raise your pages above the noise. Let's look at the major technologies you should know about.

Hypertext Markup Language (HTML and XHTML)

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, more generically, the *client*) how to display chunks of text or images.

Relegating HTML to the category of a 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.

Version 4.0 and later of the published HTML standards endeavor to define the purpose of HTML as assigning context to content, leaving the appearance to a separate standard for stylesheets. In other words, it's not HTML's role to signify that some text is italic but, rather, to signify *why* it is italic. For example, you would tag a chunk of text that conveys emphasis (via the tag) regardless of how the stylesheet or browser sets the appearance of that emphasized text.

XHTML is a more recent adaptation of HTML that adheres to stylistic conventions established by the XML (eXtensible Markup Language) standard. No new tags come with XHTML, but it reinforces the notion of tagging to denote a document's structure and content.

Cascading Style Sheets (CSS)

Specifying the look and feel of a Web page via stylesheets is a major trend taking over the modern Web. The basic idea is that given a document's structure spelled out by its HTML or XHTML, a stylesheet defines the layout, colors, fonts, and other visual characteristics to present the content. Applying a different set of CSS definitions to the same document can make it look entirely different, even though the words and images are the same.

Mastery of the fine points of CSS takes time and experimentation, but the results are worth the effort. The days of using HTML tables and transparent “spacer” images to generate elaborate multicolumn layouts are very much on the wane. Every Web developer should have a solid grounding in CSS.

Server programming

Web sites that rely on database access or change their content very frequently incorporate programming on the server that generates the HTML output for browsers and/or processes forms that site visitors fill out on the page. Even submissions from a simple login or search form ultimately trigger some server process that sends the results to your browser. Server programming takes on many guises, the names of which you may recognize from your surfing through Web development sites. PHP, ASP, .Net, JSP, and Coldfusion are among the most popular. Associated programming languages include Perl, Python, Java, C++, C#, Visual Basic, and even server-side JavaScript in some environments.

Whatever language you use, the job 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 massage the information coming from the user. Even with the new, server-based Web site design tools available, server scripting often is a task that a content-oriented HTML author will need to hand off to a more experienced programmer.

As powerful and useful as server scripting can be, it does a poor job of facilitating interactivity in a Web page. Without the help of browser scripting, each change to a page must be processed on the server, causing delays for the visitor and an extra burden on the server for simple tasks. This wastes desktop processing horsepower, especially if the process running on the server doesn't need to access big databases or other external computers.

Working together, however, server programming and browser scripting can make beautiful applications together. The pair come into play with what has become known as *Ajax* — Asynchronous JavaScript and XML. The “asynchronous” part runs in the browser, requesting XML data from, or posting data to, the server entirely in the background. XML data returned by the server can then be examined by JavaScript in the browser to update portions of the Web page. That's how many popular Web-based email user interfaces work, as well as the draggable satellite-photo closeups of Google Maps (<http://maps.google.com>).

Of helpers and plug-ins

In the early days of the World Wide Web, a browser needed to present only a few kinds of data before a user's eyes. The power to render text (tagged with HTML) and images (in popular formats such as GIF and JPEG) was built into browsers intended for desktop operating systems. Not wanting to be limited by those data types, developers worked hard to extend browsers so that data in other formats could be rendered on

the client computer. It was unlikely, however, that a browser would ever be built that could download and render, say, any of several sound-file formats.

One way to solve the problem was to allow the browser, upon recognizing an incoming file of a particular type, to launch a separate application on the client machine to render the content. As long as this helper application was installed on the client computer (and the association with the helper program was set in the browser's preferences), the browser would launch the program and send the incoming file to that program. Thus, you might have one helper application for a MIDI sound file and another for an animation file.

Beginning with Netscape Navigator 2 in early 1996, software *plug-ins* for browsers enabled developers to extend the capabilities of the browser without having to modify the browser. Unlike a helper application, a plug-in can enable external content to blend 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 (streaming) 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.

Today's browsers tend to ship with plug-ins that decode the most common sound-file types. Developers of plug-ins for Internet Explorer for the Windows operating system commonly implement plug-ins as ActiveX controls — a distinction that is important to the underpinnings of the operating system but not to the user.

Plug-ins and helpers are valuable for more than just audio and video playback. A popular helper application is *Adobe Acrobat Reader*, which displays Acrobat files that are formatted just as though they were being printed. But for interactivity, developers today frequently rely on Macromedia Corporation's *Flash* plug-in. Created using the Macromedia Flash authoring environment, a Flash document can have active clickable areas and draggable elements. Some authors even simulate artistic video games and animated stories in Flash. A browser equipped with the Flash plug-in displays the content in a rectangular area embedded within the browser page.

One potential downside for authoring interactive content in Flash or similar environments is that if the user does not have the correct plug-in version installed, it can take some time to download the plug-in (if the user even wants to bother). Moreover, once the plug-in is installed, highly graphic and interactive content can take longer to download to the client (especially on a dial-up connection) than some users are willing to wait. This is one of those situations in which you must balance your creative palette with the user's desire for your interactive content.

Another client-side technology — the Java applet — was popular for a while in the late 1990s but has fallen out of favor for a variety of reasons (some technical, some corporate–political). But this has not diminished the use of Java as a language for server and even cellular telephone programming, extending well beyond the scope of the language's founding company, Sun Microsystems.

JavaScript: A Language for All

Sun's 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 by this situation when I first read about Java's preliminary specifications in 1995. I would have preferred a language that casual programmers and scripters who were comfortable with authoring tools, such as Apple's once-formidable HyperCard and Microsoft's 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 professional 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 Communications, Inc. Born under the name *LiveScript*, this language was developed in parallel with a new version of 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 LiveScript on the server.

On the client side — in HTML documents — authors could employ scripts written in this new language to enhance Web pages in a number of ways. For example, an author could use LiveScript to make sure that the user had filled in a required text field with an e-mail address or credit card number. 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 computing horsepower to work. In essence, LiveScript could provide HTML-level interaction for the user.

LiveScript becomes JavaScript

In early December 1995, just prior to the formal release of Navigator 2, Netscape and Sun Microsystems jointly announced that the scripting language thereafter would 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 expected.

Before the announcement, the language was already related to Java in some ways. Many of the basic syntax elements of the scripting language were reminiscent of the Java style. 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 anything else on the page). Instead of Java's full-blown programming language vocabulary (and conceptually more difficult to learn object-oriented approach), JavaScript had a small vocabulary and a 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 provided a simpler way of building Java applets. To this day, some new programmers believe JavaScript is synonymous with the Java language: They post Java queries to JavaScript-specific Internet newsgroups and mailing lists.

The fact remains that client-side Java and JavaScript are more different than they are similar. The two languages employ entirely different interpreter engines to execute their lines of code.

Enter Microsoft and others

Although the JavaScript language originated at Netscape, Microsoft acknowledged the potential power and popularity of the language by implementing it (under the JScript name) in Internet Explorer 3. Even if Microsoft might prefer that the world use the VBScript (Visual Basic Script) language that it provides in the Windows versions of IE, the fact that JavaScript is available on more browsers and operating systems makes it the client-side scripter's choice for anyone who must design for a broad range of users.

With scripting firmly entrenched in the mainstream browsers from Microsoft and Netscape, newer browser makers automatically provided support for JavaScript. Therefore, you can count on fundamental scripting services in browsers such as Opera or the Apple Safari browser (the latter built upon an Open Source browser called KHTML). Not that all browsers work the same way in every detail — a significant challenge for client-side scripting that is addressed throughout this book.

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 site author. A Web designer who ignores JavaScript is akin to a plumber who bruises his knuckles by using pliers instead of the wrench from 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:

- Getting 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
- Distributing small collections of databaselike information and providing a friendly interface to that data
- Controlling multiple-frame navigation, plug-ins, or Java applets based on user choices in the HTML document
- Preprocessing data on the client before submission to a server
- Changing content and styles in modern browsers dynamically and instantly in response to user interaction

At the same time, it is equally important to understand what JavaScript is *not* capable of doing. 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 uses 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 or server computer
- Capturing live data streams from the server for retransmission
- Sending secret e-mails from Web site visitors to you

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. Neither 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.