# Chapter 1

## Learning the Fundamentals

Imagine being able to create interactive multimedia adventure games that anyone can play over the World Wide Web. Imagine being able to create animated product catalogs that not only help your customers find the products they want but enable them to purchase them using secure online payment systems. Imagine being able to create database applications for use by your company's sales force from one end of the country to another via the company's intranet. With JavaScript, you no longer have to imagine—you can do it all.

JavaScript is the powerful programming language for the Web that not only enables the development of truly interactive web pages, but also is the essential glue that integrates *HTML*, *XML*, *Java applets*, *ActiveX Controls*, *browser plug-ins*, *server scripts*, and other web *objects*, permitting developers to create *distributed applications* for use over the Internet and over corporate *intranets* as well.




Adapted from *Mastering JavaScript Premium Edition*
by James Jaworski
ISBN 0-7821-2819-X

If the terms in the preceding paragraphs are a bit confusing to you, you've come to the right place to begin your involvement with JavaScript and the world of interactive web page development. In this chapter, I will provide the background information you need to begin mastering the JavaScript language. I'll begin with the concepts that are essential to understanding the operation of the Web.

**NOTE**

**JavaScript is supported by Netscape Navigator, Microsoft Internet Explorer, Mozilla, Chimera, Sun's HotJava, Opera Software's Opera Browser, and other browsers. As such, it is an important tool for both current and future web development. Throughout the JavaScript sections of this book, we will emphasize the scripting capabilities provided by Navigator 6 and 7 (JavaScript 1.5) and Internet Explorer 6 (JScript 5.6). The other JavaScript-capable browsers take their lead from Navigator and Internet Explorer, but may not fully support all of the features of JavaScript 1.5 or JScript 5.6. For instance, the Opera 5 browser claims to support "most" of the JavaScript 1.4 core.**

# The Web

The Web is one of the most popular services provided via the Internet. At its best, it combines the appeal of exploring exotic destinations with the excitement of playing a video game, listening to a music CD, or even directing a movie, and you can do it all by means of an intuitive, easy-to-use, graphical user interface. Probably the most appealing aspect of the Web, however, is the fact that it isn't just for spectators. Once you have some experience with web *authoring tools* (and even something as simple as Notepad or SimpleText can be a web authoring tool), you can publish yourself—and offer over the Web anything you want to make available, from your company's latest research results to your own documentary on the lives of the rich and famous.

To many people, the most familiar element of the Web is the *browser*. A browser is the user's window to the Web, providing the capability to view web documents and access web-based services and applications. The most popular browsers are Netscape's Navigator and Microsoft's Internet Explorer, the last few versions of which support JavaScript. Both browsers are descendants of the Mosaic browser, which was developed by a team of programmers, notably including Marc Andreessen, at the National Center for Supercomputing Applications (NCSA), located at the University

of Illinois, Urbana-Champaign. Mosaic's slick graphical user interface (GUI, pronounced "gooey") helped transform the Web from a research tool to the global publishing medium it is today.

Today's web browsers have gone far beyond Mosaic's GUI features with multimedia capabilities and browser-based implementations of software runtime environments such as Java and JavaScript. These programming languages make it possible to develop web documents that are highly interactive, meaning they do more than simply connect you to another web page elsewhere on the Internet. *Web documents created with JavaScript contain programs*—which you, as the user of a browser, run entirely within the context of the web pages that are currently displayed. This is a major advance in web publishing technology. It means, for one thing, that you can run web-based applications without having to install additional software on your machine.

To publish a document on the Web, you must make it available to a web *server*. Web servers retrieve web documents in response to browser requests and return the documents to the requesting browsers. Web servers also provide gateways that enable browsers to access web-related applications, such as database searches and electronic payment systems.

The earliest web servers were developed by CERN and NCSA. These servers were the mainstay of the Web throughout its early years. Lately, commercial web servers, developed by Netscape, Sun Microsystems, Microsoft, and other companies, have become increasingly popular on the Web, and the open-source Apache web server is still the most widely used according to many surveys. These servers are designed for higher performance and to facilitate the development of complex web applications. They also support the development of server-based applications using languages such as Perl, Java, Visual Basic, and JavaScript. Code written in these languages can be integrated very tightly with the server, with the result that server-side programs are executed very efficiently.

Because the Web uses the Internet as its communication medium, it must follow Internet communication *protocols*. A protocol is a set of rules governing the procedures for exchanging information. The Internet's *Transmission Control Protocol* (TCP) and *Internet Protocol* (IP) enable worldwide connectivity between clients and servers. Layered atop the TCP/IP protocols for communication across the Internet, the Web also uses its own protocol, called the *Hypertext Transfer Protocol* (HTTP), for exchanges between browsers and servers. Browsers use HTTP to request documents from servers, and servers use it to return requested documents to browsers. Figure 1.1 shows an analogy between the English language

and telephony protocols over the phone system on one hand, and HTTP and TCP/IP over the Internet on the other hand. Browsers and servers communicate via HTTP over the Internet the same way an American and an Englishman would communicate via English over a phone system.

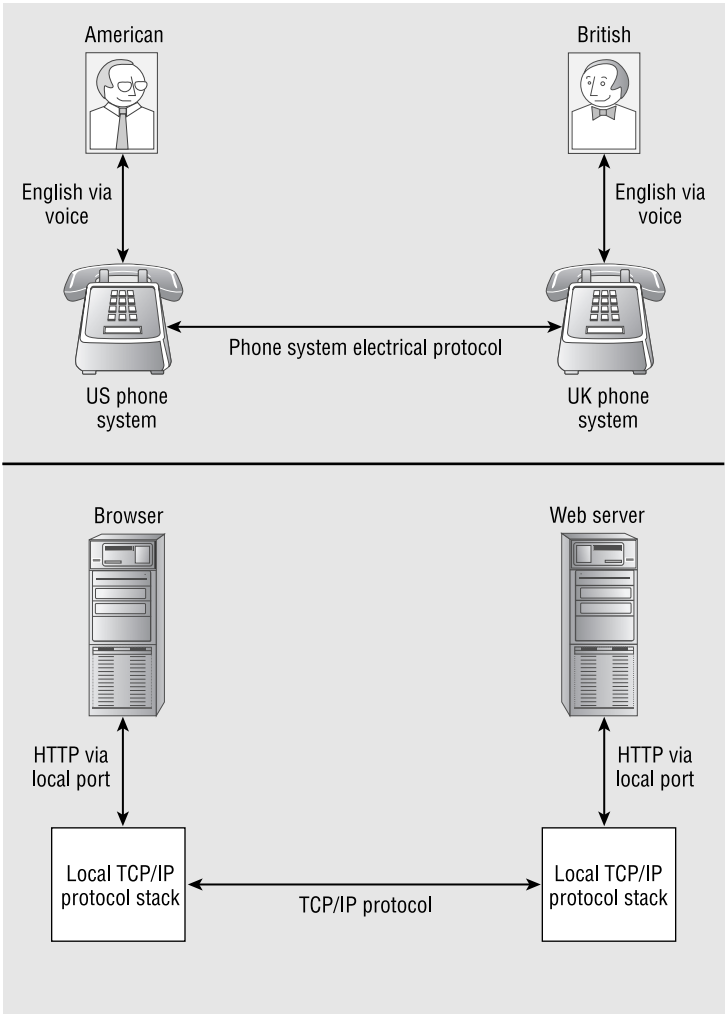


**FIGURE 1.1:** Browsers and servers communicate via HTTP over the Internet the same way an American writer and a British editor communicate via English over a phone system.

# The Hypertext Markup Language

The Hypertext Markup Language (HTML) is the *lingua franca* of the Web. It is used to create web pages and is similar to the codes used by some word processing and document layout programs.

HTML uses ordinary text files to represent web pages. The files consist of the text to be displayed and the *tags* that specify *how* the text is to be displayed. For example, the following line from an HTML file shows the text of a title between the appropriate title tags:

```
<TITLE>Mastering JavaScript</TITLE>
```

The use of tags to define the elements of a web document is referred to as *markup.* Some tags specify the title of a document; others identify headings, paragraphs, and hyperlinks. Still others are used to insert forms, images, multimedia objects, and other features in web documents.

**NOTE**

**This book assumes that you have a working knowledge of HTML. This section briefly reviews the important aspects of the language. If you have not used HTML, you should also check out the links to HTML tutorials and reference information located on this book's web page at `www.sybex.com`.**

Tags always begin with a left angle bracket (<) and end with a right angle bracket (>). The name of the tag is placed between these two symbols. Usually, but not always, tags come in pairs, to surround the text that is marked up. Such tags are referred to as *surrounding* tags. For example, HTML documents begin with the `<HTML>` tag and end with the `</HTML>` tag. The first tag of a pair of tags is the *beginning* or *opening* tag, and the second tag of the pair is the *ending* or *closing* tag. The ending tag has the same name as the beginning tag except that a / (forward slash character) immediately follows the <.

Other tags, known as *separating* tags, do not come in pairs, and have no closing tags. These tags are used to insert such things as line breaks, images, and horizontal rules within marked-up text. An example of a separating tag is `<HR>`, which inserts a horizontal rule (a line) across a web page.

Both surrounding and separating tags use *attributes* to specify properties of marked-up text. These attributes and their *attribute values*, if any, are included in the tag. For example, you can specify a horizontal rule 10 pixels high and the entire width of the browser window using the following tag:

```
<HR SIZE="10">
```

This tag contains a SIZE attribute that is assigned an attribute value of 10.

**NOTE**

**Attributes and attribute values are placed in the opening tag of a pair of surrounding tags and don't have to be repeated when the tag is closed—for example, <P ALIGN="center">info</P>.**

Listing 1.1 contains a sample HTML document that illustrates the use of tags in marking up a web page. Figure 1.2 shows how Netscape Navigator displays this HTML document. The <HTML> and </HTML> tags identify the beginning and end of the HTML document. The document contains a head, identified by the <HEAD> and </HEAD> tags, and a body, identified by the <BODY> and </BODY> tags. The document's head contains a title that is marked by the <TITLE> and </TITLE> tags. (The title appears at the top of the Navigator window.)

**NOTE**

**You can download the file for Listing 1.1, ch01-01.htm, from the Sybex website, on the product page for this book.**

**Listing 1.1: Example HTML Document (ch01-01.htm)**

```
<HTML>
<HEAD>
<TITLE>This text is the document's title.</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="CENTER">This is a centered heading.</H1>
<P>This is the first paragraph.</P>
<P>This is the second paragraph.</P>
<HR SIZE="10">
<P ALIGN="CENTER">This paragraph is centered and
  below the horizontal rule.</P></BODY>
</HTML>
```

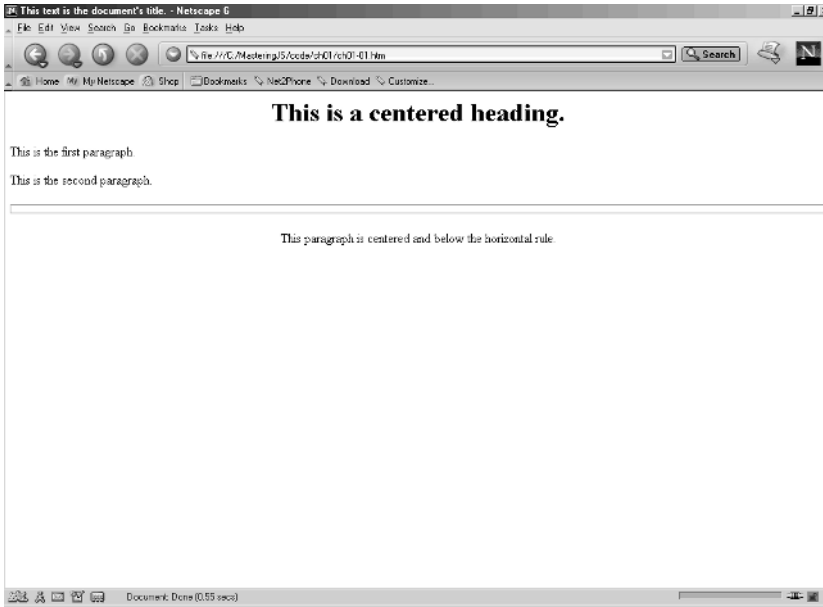


**FIGURE 1.2:** A browser display of the HTML document shown in Listing 1.1

Here are a few items to notice in this listing:

- The document's body contains a Heading 1 that is marked by the `<H1>` and `</H1>` tags. The opening `<H1>` tag uses the `ALIGN` attribute to center the heading.
- Two paragraphs immediately follow the heading. These paragraphs are marked by the paragraph tags `<P>` and `</P>`.
- Following these two paragraphs is a horizontal rule with its `SIZE` attribute set to 10.
- The last element of the document's body is a paragraph that uses the `ALIGN` attribute to center the paragraph.

# The Development of HTML and XHTML

HTML was originally developed by Tim Berners-Lee at CERN. Since then, it has evolved through several major revisions. Each revision adds new tags that increase the expressive power of the language. For example, HTML 2 added the capability to include forms within web documents, and HTML 3.2 added tags for tables and tags that support the use of JavaScript and Java.


As of this writing, HTML 4.01 is the latest official version of the HTML language; and a hybrid of XML and HTML, called Extensible Hypertext Markup Language (XHTML), is beginning to gain popularity. HTML 4 adds support for international text, greater accessibility, more flexible tables, generic objects, printing, and advanced style sheets.

Although HTML is periodically standardized, the language continues to grow as the result of new tags, attributes, and attribute values that browser developers introduce. Because Netscape and Microsoft hold the largest share of the browser market, they have taken the lead in defining new additions to HTML. These additions are not part of the official HTML language, so they are referred to as *extensions*. Most extensions are eventually integrated into the official version of HTML.

Although HTML 4 is the current standard, some believe its days are numbered. XHTML was released as a recommendation by the World Wide Web Consortium (W3C) in January 2000. XHTML is essentially a simple reformulation of HTML to be more like XML, the *Extensible Markup Language*. Simplicity and extensibility are XHTML's primary advantages over HTML. XHTML removes the flexible coding supported by HTML. This makes XHTML simpler and easier to parse, allowing XHTML parsers to be quicker and smaller. Because XHTML is an XML application, it is easily extended. New tags and attributes can be defined and added to those that are defined in the standard. Much of XHTML is identical to HTML, however, and the changes—to force paired tags, for example—are pretty easy: The <HR> tag referenced earlier would be written as `<hr size="10" />`. Notice that XHTML tag and attribute names are always lowercase, and that all attribute values must be in quotes.

Even though XHTML is the logical successor to HTML, there is no need to convert all your web pages to the new standard. If you do, you'll find that some of your pages won't be rendered correctly by non-XHTML capable browsers. In addition, the document object models supported by current browsers are HTML based, although theoretically they should also work with XHTML implementations. Even though Navigator 6 and 7 and Internet Explorer 5 provide XML support, their primary capabilities and features still center around HTML.

# Cascading Style Sheets

*Style sheets* provide the capability to control the way HTML elements are laid out and displayed. For example, you can use style sheets to control the color, font, and spacing used with different HTML elements. Support

for Cascading Style Sheets (CSS) was developed by the W3C and introduced with HTML 3.2, and additional CSS support was added in HTML 4. *Cascading* refers to the capability to use multiple levels of style sheets for a document where one level of style can be used to define another.

Two levels of CSS have been defined. CSS1 is a simple style sheet mechanism that allows basic styles (for example, fonts, colors, and spacing) to be associated with HTML elements. CSS1 is an outgrowth of HTML 3.2 and is supported by Internet Explorer 3 (and later), Navigator 4 (and later), and other browsers. CSS2 builds on CSS1 to add support for media-specific style sheets, content positioning, downloadable fonts, table layout, internationalization, automatic counters and numbering, and other capabilities.

In addition to CSS1 and CSS2, Navigator 4 introduced JavaScript Style Sheets (JSS). JSS is similar to CSS1 and makes styles available as JavaScript properties, although few developers, if any, use JSS, given the predominance of Internet Explorer on the Web.

# Helper Applications

Most graphical web browsers provide support for viewing images in common graphics formats, such as Graphics Interchange Format (GIF) and Joint Photographic Experts Group (JPEG). Some can even play audio files. However, most browsers do not provide much more than that in terms of multimedia features. Instead of building larger, more complicated browsers to handle many different file formats, browser developers use *helper applications*. When a browser encounters a file type that it does not know how to handle, it searches its list of helper applications to see if it has one that can deal with the file. If a suitable helper is found, then the browser executes the helper and passes it the name of the file to be run. If an appropriate helper cannot be found, then the browser prompts the user to identify which helper to use or to save the file for later display.

## External Viewers and Plug-Ins

Early helper programs operated independently of the web browser. These programs, referred to as *external viewers*, were executed separate from the browser and created their own windows to display various types of files. Netscape and Microsoft developed the capability for their second-generation browsers to use *plug-in* or *add-in modules*, which not only execute automatically when needed but display their output in the browser window. Since then, numerous companies have developed plug-in modules

to support everything from the three-dimensional worlds created by the Virtual Reality Modeling Language (VRML) to CD-quality audio and a variety of streaming video formats.

Plug-in modules are generally quicker to load and more efficient than external viewers. Because they execute with the browser, they can be accessed from within the browser environment. Netscape lets you control plug-in modules from Java and JavaScript code via its LiveConnect toolkit. Microsoft provides a similar capability through its Internet Explorer Object Model.

# Using MIME Types to Identify Helpers for File Formats

So far, I've described how browsers use helper applications to display different types of files, but how does a browser know which helpers to use for a given file? The answer lies in MIME types and sometimes in filename suffixes.

Multipurpose Internet Mail Extensions (MIME) was originally developed as a standard for including different types of files in electronic mail. It was subsequently adopted for web servers and browsers to identify the types of files referenced in a web page.

MIME identifies file types using a *type/subtype* naming scheme. Examples of common MIME types are text/plain, text/html, image/gif, and video/quicktime. The first component of a MIME type identifies the general type of a file, and the second part identifies the specific type within the general category. For example, the text/plain and text/html types both belong to the text category, but they differ in their subtypes. Table 1.1 lists some common MIME types.

**TABLE 1.1:** Example MIME Types

| MIME Type | Description |
|---|---|
| text/plain | Generic ASCII text file |
| text/html | Text file containing HTML |
| image/gif | Image in Graphics Interchange Format |
| image/jpeg | Image in Joint Photographic Experts Group format |
| audio/x-wav | File containing sounds stored in the Windows audio file format |

**TABLE 1.1 continued:** Example MIME Types

| MIME Type | Description |
|---|---|
| video/mpeg | Video in the Moving Pictures Experts Group format |
| video/quicktime | Video in the Apple QuickTime format |
| application/octet-stream | Raw (unformatted) stream of bytes |
| application/x-javascript | File containing JavaScript source code |

Web servers contain configuration files that match file extensions with their MIME types. For example, files that end with the extension `.htm` or `.html` are associated with the text/html MIME type, and files that end with `.jpg`, `.jpe`, or `.jpeg` are associated with the image/jpeg MIME type.

Browsers also contain configuration information about MIME types. This information is used to map MIME types to the helper application that displays files of that type.

When a browser requests a file from a web server, the server uses the filename's extension to look up the file's MIME type if it's not already specified by the program generating the material. The server can also try to guess the type of file from its contents if there is no filename extension, or simply assign a MIME type if the file is being generated by the server on-the-fly. The server then identifies the file's MIME type to the browser. The browser uses the file's MIME type to determine which helper application, if any, is to be used to display the file. If the file is to be displayed by an external viewer, the browser waits until the file has been completely received before launching the viewer. If the file is to be displayed by a plug-in, the browser launches the plug-in and passes the file to the plug-in as the file is received. This process enables the plug-in to begin displaying the file before it is fully loaded (or *stream* it), which is an important capability of audio- and video-intensive applications.

# Uniform Resource Locators (URLs)

A *Uniform Resource Locator (URL)* is the notation used to specify the address of an Internet file or service.

A URL always contains a *protocol identifier,* such as `http` or `ftp`, and often a host name, such as `home.netscape.com`, `www.microsoft.com`, and `ftp.cdrom.com`, which appear in the previous examples. The most commonly used protocol identifiers are `http` and `ftp`, but if you examine the Protocol Helpers section of your browser's Preferences menu (sometimes reached from Tools or another top level menu), you will find support for older, more obscure identifiers such as `wais` and `gopher`. The protocol identifier is also referred to as a *scheme*. When you write a web (HTTP) URL, the protocol identifier is followed by `://` and then the host name of the computer to which the protocol applies. (In URLs, pathnames are written using forward slash [`/`] characters rather than backslash [`\`] characters.) For example, to access the main home page of Microsoft on the host named `www.microsoft.com`, you would use the URL `http://www.microsoft.com`. To access the root directory of the File Transfer Protocol (FTP) server hosted by `ftp.cdrom.com`, you would use the URL `ftp://ftp.cdrom.com`.

In addition to the host name, the URL can specify the pathname and filename of a file to be accessed by adding a single `/` character followed by the name. For example, the Internet book area on the Sybex website is located in the `Internet` subdirectory of the directory `sybexbooks.nsf` on Sybex's web server's root directory. The URL for this area is therefore `http://www.sybex.com/sybexbooks.nsf/Internet/`.

**NOTE**

**URLs can also contain additional addressing components, such as a port name before the path and filename and a file offset after the filename.**

**THE *FILE* PROTOCOL IN URLS**

Your browser can use the `file` protocol to access files located on your local machine. Suppose the file `test.htm` was located on your Windows desktop. The path to this file would be `c:\windows\desktop\test.htm`. To open the file with your browser, you would use the following URL: `file://localhost/C|/WINDOWS/Desktop/test.htm`.

The host name `localhost` in the previous URL refers to the local filesystem and can be omitted safely. However, you should retain

CONTINUED ➞

the slash following `localhost`. The previous URL could be thus be written as follows: `file:///C|/WINDOWS/Desktop/test.htm`.

Note that in both examples the `C:` drive designation is written as `C|` instead. If you are using a Macintosh or Unix browser, this format might vary slightly. On a Mac, for example, a file reference might appear more akin to `file://localhost/Users/demo/Desktop/test.htm`.

# The Hypertext Transfer Protocol (HTTP)

HTTP is the protocol used for communication between browsers and web servers. HTTP uses a request/response model of communication. A browser establishes a connection with a server and sends a URL request to the server. The server processes the browser's request and sends a response back to the browser.

A browser connects with a web server by establishing a TCP connection, by default at port 80 of the server. You can specify server ports other than 80; for instance, to connect to port 8234 on the `www.fictionalhost.com` server, the URL would be `http://www.fictionalhost.com:8234/`. This port is the address at which web servers "listen" for browser requests. Once a connection has been established, a browser sends a request to the server. This request specifies a request method; the URL of the document, program, or other resource being requested; the HTTP version being used by the browser; and other information related to the request.

Several request methods are available. `GET`, `HEAD`, and `POST` are the most commonly used:

***GET*** Retrieves the information contained at the specified URL. You can also use this method to *submit* data collected in an HTML *form* (the topic of Chapter 5, "Processing Forms") or to invoke a Common Gateway Interface (CGI) program (discussed in the next section). When the server processes a `GET` request, it delivers the requested information (if it can be found). The server inserts at the front of the information an HTTP header that provides data about the server, identifies any errors that

occurred in processing the request, and describes the type of information being returned as a result.

***HEAD*** Similar to the GET method, except that when a web server processes a HEAD request, it returns only the HTTP header data and not the information that was the object of the request. The HEAD method is used to retrieve information about a URL without actually obtaining the information addressed by the URL.

***POST*** Informs the server that the information appended to the request is to be sent to the specified URL. The POST method is typically used to send form data and other information to CGI programs. The web server responds to a POST request by sending back header data followed by any information generated by the CGI program as a result of processing the request.


The current version of HTTP is HTTP 1.1. It incorporates performance, security, and other improvements to the original HTTP 1. A new version of HTTP, referred to as HTTP-NG, is currently being defined. (The *NG* stands for *next generation*.) The goal of HTTP-NG is to simplify HTTP and make it more extensible. However, little progress has taken place over the past few years, and the project may be considered dead.

# Common Gateway Interface Programs

The *Common Gateway Interface* (CGI) is a standard that specifies how web servers can use external programs. Programs that adhere to the CGI standard are referred to as *CGI programs*. These programs can be used to process data submitted with forms, to perform database searches, and to support other types of web applications, such as clickable image maps.

A browser request for the URL of a CGI program comes about as the result of a user clicking a link, requesting the output of a CGI program (for example, many sites have their default home page generated by a CGI program rather than as a static HTML page), or submitting a form. The browser uses HTTP to make the request. When a web server receives the request, the web server executes the CGI program and also passes it any data that was submitted by the browser. When the CGI program performs its processing, it usually generates data in the form

of a web page, which it returns via the web server to the requesting browser.

The CGI standard specifies how data may be passed from web servers to CGI programs and how data should be returned from CGI programs to the web server. Table 1.2 summarizes these interfaces. In Chapter 5 and Chapter 7, "Interfacing JavaScript with CGI Programs," you'll study CGI and learn how to create CGI programs.

**TABLE 1.2:** CGI Summary

| Method of Communicating | Interface | Description |
|---|---|---|
| Command-line arguments | Web server to CGI program | Data is passed to the CGI program via the command line that is used to execute the program. Command-line arguments are passed to CGI programs as the result of `ISINDEX` queries. |
| Environment variables | Web server to CGI program | A web server passes data to the CGI program by setting special *environment variables* that are available to the CGI program via its environment |
| Standard input stream | Web server to CGI program | A web server passes data to a CGI program by sending the data to the standard character input stream associated with the CGI program. The CGI program reads the data as if a user manually entered it at a character terminal. |
| Standard output stream | CGI program to web server | The CGI program passes data back to the web server by writing the data to its standard output stream. The web server intercepts this data and sends it back to the browser that made the CGI request. |

# Java Applets

The Java language, developed by Sun Microsystems, Inc., has realized tremendous popularity. Although it was originally developed as a language for programming consumer electronic devices, Java has increasingly been adopted as a hardware- and software-independent platform for developing

advanced web applications. Java can be used to write stand-alone applications, but a major reason for its popularity is that you can also develop Java programs that can be executed by a web browser.

Java programs that can be executed by the web browser are called *applets* rather than applications, because they cannot be run outside the browser's window without a separate viewer or helper application. (*Application* usually implies a complete, stand-alone program.) Programmers create Java applets using built-in programming features of the Java Developer's Kit (JDK). Web pages, written in HTML, reference Java applets using the <APPLET> or <OBJECT> tag, much as images are referenced using the <IMG> tag. When a browser loads a web page that references a Java applet, the browser requests the applet code from the web server. When the browser receives the applet code, it executes the code and allocates a fixed area of the browser window. This area is identified by attributes specified with the <APPLET> tag. The applet is not allowed to update the browser display or handle events outside its allocated window area.

By way of comparison, JavaScript provides access to the entire web page, but is a much smaller, lighter-weight programming language that also doesn't support many of the more advanced object-oriented programming features of Java. Netscape Navigator and Microsoft Internet Explorer provide the capability for JavaScript scripts to load Java applets, access Java objects, and invoke their methods.

# ActiveX—Microsoft Objects

ActiveX is Microsoft's approach to executing objects other than Java applets in Internet Explorer. The name *ActiveX* was used to make it seem like a new and innovative technology. However, ActiveX is nothing more than Component Object Model (COM) objects that can be downloaded and executed by Internet Explorer. COM traces its origin back to the Object Linking and Embedding (OLE) technology of Microsoft Windows 3.1.

*COM objects* are instances of *classes* (object types) that are also organized into *interfaces*. Each interface consists of a collection of *methods* (functions). COM objects are implemented inside a *server* (dynamic-link libraries, operating system service, or independent process) and are accessed via their methods. The *COM library* provides a directory of available COM objects. Over the years since Windows 3.1, many software components have been developed as COM objects.

ActiveX components are COM objects that implement a specific type of interface. They are important in that they provide a means for the large base of COM objects to be reused within Internet Explorer. They also allow older languages, such as C++ and C, to be used to build components for web applications.

Although ActiveX components allow the use of legacy software in Internet Explorer, they also present some drawbacks. The most significant drawback is that ActiveX is only supported by Internet Explorer 4 and later—no other browser (including earlier versions of Internet Explorer) can use ActiveX. ActiveX has also been criticized for its poor security. An ActiveX component is not required to behave in a secure manner like a Java applet or JavaScript script. In fact, it has been demonstrated that ActiveX components can be used to steal or modify sensitive information or completely wipe out a user's system. Microsoft has countered this vulnerability by allowing ActiveX components to be digitally signed. This does not prevent ActiveX components from violating security, but, in some cases, a signature can be used to determine whether a particular website is responsible for causing damage.

ActiveX components are useful in intranet applications where all users of a particular company are required to use Internet Explorer and the components are signed by the company or a trusted developer. Because the Internet Explorer Object Model allows ActiveX components to be accessed from JavaScript, JavaScript scripts can be used to integrate the ActiveX components into the intranet applications.

# A Brief History of JavaScript

Often, one programming language evolves from another. For example, Java evolved from C++, which evolved from C, which evolved from other languages. Similarly, Netscape originally developed a language called *LiveScript* to add a basic scripting capability to both Navigator and its web-server line of products; when it added support for Java applets in its release of Navigator 2, Netscape replaced LiveScript with JavaScript. Although the initial version of JavaScript was little more than LiveScript renamed, JavaScript has been subsequently standardized through the European Computer Manufacturing Association (ECMA) and is now also referred to as ECMAScript (formally ECMA-262).

**NOTE**

**Although JavaScript bears the name of Java, JavaScript is a very different language that is used for a very different purpose.**

JavaScript supports both web browser and server scripting. Browser scripts are used to create dynamic web pages that are more interactive, more responsive, and more tightly integrated with plug-ins, ActiveX components, and Java applets. JavaScript supports these features by providing special programming capabilities, such as the ability to dynamically generate HTML and to define custom event-handling functions.

You include JavaScript scripts in HTML documents via the HTML <SCRIPT> tag. When a JavaScript-capable browser loads an HTML document containing scripts, it evaluates the scripts as they are encountered. The scripts may be used to create HTML elements that are added to the displayed document or to define functions, called *event handlers,* that respond to user actions, such as mouse clicks and keyboard entries. Scripts can also be used to control plug-ins, ActiveX components, and Java applets.

Microsoft implemented its version of JavaScript, named JScript, in Internet Explorer 3. The scripting capability of Internet Explorer 3 is roughly equivalent to Navigator 2. Netscape introduced JavaScript 1.1 with Navigator 3 and JavaScript 1.2 with Navigator 4. JavaScript 1.1 added a number of new features, including support for more browser objects and user-defined functions. JavaScript 1.2 added new objects, methods, properties, and support for style sheets, layers, regular expressions, and signed scripts.

Microsoft introduced its ECMAScript-compliant version of JScript in Internet Explorer 4. JScript is tightly coupled to Internet Explorer and allows almost all HTML elements to be scripted. Microsoft also included server-side JavaScript support with its Internet Information Server (IIS). It later developed a more general approach to server-side scripting with its Windows Script Host and remote scripting technologies. Remote scripting allows Internet Explorer to remotely execute scripts on a server and receive the server script outputs within the context of a single web page.

Netscape and Microsoft submitted their scripting languages to the ECMA for standardization. ECMA released the Standard ECMA-262 in June of 1997. This standard describes the ECMAScript language, which is a consolidation of the core features of JavaScript and JScript.

Updated versions of this standard were released in June 1998 (Revision 2) and December 1999 (Revision 3). ECMA also released ECMA-290 in June 1999. ECMA-290 covers the development of reusable components in ECMAScript.

Microsoft worked closely with the ECMA and updated Internet Explorer 4 and JScript (JScript 3.1) to achieve ECMAScript compliance. Navigator achieved ECMAScript compliance with JavaScript 1.3, which is supported in Navigator 4.06 through 4.7.

Internet Explorer 5 introduced JScript 5, which provides additional scripting capabilities, such as the `try - catch` statement. This statement provides advanced error handling support and is included in ECMAScript Revision 3. Internet Explorer 5.5 was introduced after ECMAScript Revision 3 and provides full Revision 3 support. Navigator 6.0 and later supports JavaScript 1.5, which is fully compliant with ECMAScript Revision 3.

While Netscape and Microsoft were busy introducing new versions of their browsers and scripting languages, Opera Software (`www.opera-software.com`) launched another JavaScript-compatible browser. In addition, Sun jumped into the JavaScript field with its HotJava browser. HotJava 3.0 is ECMAScript compliant. Other browser developers followed by developing JavaScript-capable browsers of their own.

Another JavaScript-related standardization effort was initiated by the W3C to standardize the basic objects made available by browsers when processing HTML and XML documents. This effort resulted in a specification known as the Document Object Model (DOM) Level 1. It provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. The DOM is like an application programming interface (API) for HTML and XML documents. However, the DOM is not a complete API, in that it does not specify the events that occur when a user interacts with an HTML or XML document (and methods for handling them). Version 6 and 7 of Navigator and version 5 of Internet Explorer support the DOM.

Today, the latest version of JavaScript is 1.5, but the additions to JavaScript in version 1.4 and 1.5 are unlikely to influence your day-to-day web development: Runtime errors are reported differently, regular expressions have been enhanced, functions can be conditionally declared, and named read-only constants are now supported. Netscape Navigator 6 and later, Microsoft Internet Explorer 6.0 and later, and Mozilla all support JavaScript 1.5.

# Java Servlets and JavaServer Pages

Sun Microsystems developed the Java Servlet API as an extension to the standard Java specification; it provides a way to write modules that run within a server to handle requests in a client-server architecture. You can think of servlets as applets that run on the server side rather than the client side. The fundamental Servlet API isn't tied to the HTTP protocol, but the API does have a framework specifically tailored to handling HTTP requests. A number of different vendors have products that implement the Servlet API, so if you write a web-based application using it, you won't necessarily be tied to a single vendor's products. Although there can be performance benefits to using servlets over a typical CGI script, perhaps the most attractive feature of using servlets is the full access to the rest of Java's standard APIs (for instance, the JDBC API, which provides a standard interface for connection to a variety of SQL database stores). If your project will reuse or interface with other modules of Java code, you should consider using this technology instead of standard CGI techniques.

JavaServer Pages (JSP) technology is an extension to Servlets that specifies ways to dynamically author HTML and XML pages. JSPs are particularly suited to situations where you need to change certain aspects of a page's content but can use a template to provide the basic format and structure of the page. For instance, an application that needs to display invoices online to users might use an Invoice template that defines fonts, tables, and headers and footers, but might rely on application logic to fill in the line items and dollar amounts.

**NOTE**

**The Tomcat Server is an open-source reference implementation of the Servlet and JSP technologies that runs on Windows and a variety of Unix platforms. You can download it free from** `http://jakarta.apache.org/tomcat/`.

# ASP, Windows Scripting Host, and Remote Scripting

Microsoft's Active Server Pages (ASP) is a server-side scripting environment that is similar to JSP. You can use it to include server-side scripts and ActiveX components with HTML pages. The combined HTML and script

file is stored as an ASP file. When a browser requests the ASP file from your web server, the server invokes the ASP processor. The ASP processor reads the requested file, executes any script commands, and sends the processed results as a web page to the browser. ASP pages can also invoke ActiveX components to perform tasks, such as accessing a database or performing an electronic commerce transaction. Because ASP scripts run on the web server and send standard HTML to the browser, ASP is browser independent.

Microsoft introduced ASP with IIS version 3. It also works with later versions of IIS, Personal Web Server for Windows 95, and Peer Web Server for Windows NT Workstation.

As a result of the success of ASP, Microsoft developed Windows Script Host (WSH), a technology that allows scripts to be run on Windows 95, 98, ME, NT 4, 2000, and Windows XP. WSH is language independent and supports JScript, VBScript, and other languages. It lets you execute scripts from the Windows desktop or a console (MS DOS) window. WSH scripts are complete in themselves and do not need to be embedded in an HTML document. WSH is an exciting technology in that it extends the capabilities of JScript beyond the Web to the Windows desktop and operating system. You can use WSH scripts to replace MS DOS scripts and take full advantage of the Windows GUI, ActiveX, and operating system functions in JScript scripts.

**NOTE**

**WSH can be freely downloaded from Microsoft's website at** `http://msdn.microsoft.com/scripting/`.

**NOTE**

**If you want to use a web server other than Microsoft's IIS, you can use the Sun ONE Active Server Pages component (**`http://wwws.sun.com/software/chilisoft/`**) to deploy ASP on the Apache, Sun ONE, or Zeus web server on a variety of non-Windows platforms, including Linux.**

Microsoft's latest addition to scripting technology is referred to as *remote scripting.* Remote scripting enables client-side scripts running on Internet Explorer to execute server-side scripts, running on IIS. Internet Explorer and IIS can perform simultaneous processing and communicate with each other within the context of a web page, allowing

the page to be dynamically updated with server information without having to be reloaded. This process frees the user from having to reload a web page during the execution of a web application and provides for a higher degree of interaction between the browser and web server. For example, with remote scripting, a web server can validate form data and provide the user with feedback while the user is still filling out the form.

Remote scripting allows browser/server communication to be accomplished in either a synchronous or asynchronous manner. When synchronous communication is used, a client-side script executes a server-side script and waits for the server-side script to return its result. When asynchronous communication is used, the client-side script executes the server-side script and then continues with its processing without waiting for the server-side script to finish. You can find more information about remote scripting at Microsoft's Developer Network site: `http://msdn.microsoft.com/scripting/`.

### ANOTHER SERVER-SIDE SCRIPTING SOLUTION: PHP

In addition to the popular ASP solution from Microsoft, an alternative server-side solution is offered by PHP, an open-source solution that is included with the Apache web server, among others. There are lots of good online references to PHP, but the best place to start is `http://www.php.net/`.

# XML AND XSL

One of the most powerful features of Navigator 6 and 7 and Internet Explorer 5 and 6 is their support for the Extensible Markup Language (XML). These browsers can display XML files directly. Moreover, they allow XML files to be scripted using JavaScript and JScript much as HTML files are scripted.

XML documents are similar to HTML documents in their use of tags and attributes to mark up text. However, XML differs from HTML in that it does not define a fixed set of markup tags. Instead, XML lets you define the tags and attributes of customized markup languages. For example, you could use XML to define a product catalog and then display the catalog directly with an XML-capable browser. You could customize the way the

catalog is displayed using CSS or the Extensible Style Language (XSL). You could also translate the XML to HTML in a format specified by an XSL style sheet.

**NOTE**

**The XML 1.0 specification is available at `www.w3.org/TR/REC-xml`. The XSL specification is available at `www.w3.org/TR/xsl/`. The XSL Transformations specification is available at `www.w3.org/TR/xslt`.**

XSL is to XML as CSS is to HTML. XSL is a language for expressing style sheets. It is organized into two parts: the XSL Transformations language (XSLT) and a vocabulary (expressed in XML) for specifying formatting semantics. XSLT lets you specify how an XML document of one type can be transformed into a document with another set of markup tags. XSLT can also be used to specify how XML documents should be translated into HTML. The second part of XSL, the formatting language, lets you specify how XML documents should be rendered for a variety of display media, such as the Web and printed documents.

**NOTE**

**XML documents can also be formatted using CSS.**

# Intranets, Extranets, and Distributed Applications

For the last few years, corporations have been deploying pure TCP/IP networks internally to take advantage of the full range of standards-based services provided by the Internet. These "company-internal internets" have become known as *intranets*. Intranets may be private networks that are physically separate from the Internet, internal networks that are separated from the Internet by a firewall, or simply a company's internal extension of the Internet.

Companies deploy intranets so that they can make internal services available to their workers using popular Internet tools and technologies. E-mail, web browsing, and web publishing are the most popular of these services. Many companies make web servers available for their employees'

intranet publishing needs. These intranet web servers allow departments, groups, and individuals within a company to conveniently share information while usually limiting access to the information published on the intranet to company employees.

The popularity of intranets as a way of communicating and of sharing information within a company has brought about a demand for more powerful and sophisticated intranet applications. The eventual goal is for the intranet to provide a common application framework from which a company's core information processing functions can be implemented and accessed. Sun, Microsoft, and other web software providers are focusing on the intranet as the primary application framework for the development of business software.

Because of its client/server architecture and user-friendly browser software, the Web is the perfect model for implementing these common intranet application frameworks. The approach taken by Netscape, Microsoft, and other web software developers is to use the web browser as the primary interface by which users connect to the intranet and run intranet and extranet applications. These applications are referred to as *distributed applications,* because their execution is distributed in part on the browser (via JavaScript, Java, ActiveX, XML, and other languages), in part on the server (via CGI programs and JavaScript and Java server-side programs), and in part on database and other enterprise servers.

Distributed intranet and extranet applications use HTML, JavaScript, Java, XML, and other languages for programming the browser-based user interface portion of the distributed application. They also use Perl, Java, Visual Basic, and other languages to perform server-side programming.

In some distributed application development approaches, Java is seen as a key technology for developing the components of distributed applications, and JavaScript is seen as the essential glue that combines these components into fully distributed web-based intranet and extranet applications. Other approaches rely less on JavaScript and more on Java.

# What's Next

This chapter covered the concepts that are essential to understanding the operation of the Web. You learned about web development languages such as HTML, XML, Java, and JavaScript. You also have been introduced to related web technologies such as HTTP, CGI, Java Servlets, JSP, ASP,

and remote scripting. You should have a basic understanding of how these elements work together when you're developing web applications.

In Chapter 2, you'll begin the exciting process of learning to use JavaScript to write sample client-side scripts. You'll begin doing some actual programming using JavaScript. If you've never done any programming, you should read the material carefully—it introduces fundamental programming concepts that are used throughout this book.