1

What Is JavaScript?

When JavaScript first appeared in 1995, its main purpose was to handle some of the input validation that had previously been left to server-side languages such as Perl. Prior to that time, a round trip to the server was needed to determine if a required field had been left blank or an entered value was invalid. Netscape Navigator sought to change that with the introduction of JavaScript. The capability to handle some basic validation on the client was an exciting new feature at a time when use of telephone modems (operating at 28.8 kbps) was widespread. Such slow speeds turned every trip to the server into an exercise in patience.

Since that time, JavaScript has grown into an important feature of every major Web browser on the market. No longer bound to simple data validation, JavaScript now interacts with nearly all aspects of the browser window and its contents. Even Microsoft, with its own client-side scripting language called VBScript, ended up including its own JavaScript implementation in Internet Explorer from its very earliest version.

In this chapter, you will learn how and why JavaScript came about, from its humble beginnings to its modern-day, feature-packed implementations. To be able to use JavaScript to its full potential, it is important to understand its nature, history, and limitations. Specifically, this chapter examines:

- ❑ The origins of JavaScript and client-side scripting
- ❑ The different parts of the JavaScript language
- ❑ The standards related to JavaScript
- ❑ JavaScript support in popular Web browsers

A Short History

Around 1992, a company called Nombas began developing an embedded scripting language called C-minus-minus (Cmm for short). The idea behind Cmm was simple: a scripting language powerful enough to replace macros, but still similar enough to C (and C++) that developers could learn it quickly. This scripting language was packaged in a shareware product called CEnvi, which first exposed the power of such languages to developers. Nombas eventually changed the name Cmm to ScriptEase because the latter sounded "too negative" and the letter C "frightened people" (`http://www.nombas.com/us/scripting/history.htm`). ScriptEase is now the driving force behind Nombas products. When the popularity of Netscape Navigator started peaking, Nombas developed a version of CEnvi that could be embedded into Web pages. These early experiments were called *Espresso Pages*, and they represented the first client-side scripting language used on the World Wide Web. Little did Nombas know that its ideas would become an important foundation for the Internet.

As Web surfing gained popularity, a gradual demand for client-side scripting languages developed. At the time, most Internet users were connecting over a 28.8 kbps modem even though Web pages were growing in size and complexity. Adding to users' pain was the large number of round-trips to the server required for simple form validation. Imagine filling out a form, clicking the Submit button, waiting 30 seconds for processing, and then being met with a message telling you that you forgot to complete a required field. Netscape, at that time on the cutting edge of technological innovation, began seriously considering the development of a client-side scripting language to handle simple processing.

Brendan Eich, who worked for Netscape at the time, began developing a scripting language called LiveScript for the upcoming release of Netscape Navigator 2.0 in 1995, with the intention of using it both in the browser and on the server (where it was to be called LiveWire). Netscape entered into a development alliance with Sun Microsystems to complete the implementation of LiveScript in time for release. Just before Netscape Navigator 2.0 was officially released, Netscape changed the name to JavaScript in order to capitalize on Java as a new Internet buzzword. Netscape's gamble paid off and JavaScript became a must-have from that point on.

Because JavaScript 1.0 was such a hit, Netscape released version 1.1 in Netscape Navigator 3.0. Right around that time, Microsoft decided to throw its hat into the ring and released Internet Explorer 3.0 with a JavaScript-clone called JScript (so-called in order to avoid any possible licensing issues with Netscape). This major step for Microsoft into the realm of Web browsers is now a date that lives in infamy for Netscape, but it also represented a major step in the development of JavaScript as a language.

After Microsoft threw its hat into the ring, three different JavaScript versions were floating around: JavaScript in Netscape Navigator, JScript in Internet Explorer, and CEnvi in ScriptEase. Unlike C and many other programming languages, JavaScript had no standards governing its syntax or features, and the three different versions only highlighted this problem. With industry fears mounting, it was decided that the language must be standardized.

In 1997, JavaScript 1.1 was submitted to the European Computer Manufacturers Association (ECMA) as a proposal. Technical Committee #39 (TC39) was assigned to "standardize the syntax and semantics of a general purpose, cross-platform, vendor-neutral scripting language" (`http://www.ecma-international.org/memento/TC39.htm`). Made up of programmers from Netscape, Sun, Microsoft, Borland, and other companies with interest in the future of scripting, TC39 met for months to hammer out ECMA-262, a standard defining a new scripting language named ECMAScript.

The following year, the International Organization for Standardization and International Electrotechnical Commission (ISO/IEC) also adopted ECMAScript as a standard (ISO/IEC-16262). Since that time, Web browsers have tried, with varying degrees of success and failure, to use ECMAScript as a basis for their JavaScript implementations.

JavaScript Implementations

Although ECMAScript is an important standard, it is not the only part of JavaScript, and certainly not the only part that has been standardized. Indeed, a complete JavaScript implementation is made up of three distinct parts (see Figure 1-1):

- ❑ The Core (ECMAScript)
- ❑ The Document Object Model (DOM)
- ❑ The Browser Object Model (BOM)

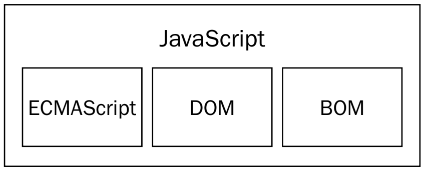

Figure 1-1

ECMAScript

ECMAScript doesn't have ties to any browser in particular and, actually, has no methods for user input or output to speak of. (It is not unlike languages such as C, which rely on external libraries to accomplish such tasks.) So what is ECMAScript? ECMA-262 (p. 2) describes it like this:

> *"ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified...apart from any particular host environment."*

A Web browser is considered a host environment for ECMAScript, but it is not the only host environment. Indeed, numerous other environments (such as Nombas's ScriptEase and Macromedia's ActionScript, used in both Flash and Director MX) can host ECMAScript implementations. So what does ECMAScript specify outside of a browser? To put it simply, ECMAScript describes the following:

- ❑ Syntax
- ❑ Types
- ❑ Statements
- ❑ Keywords
- ❑ Reserved Words
- ❑ Operators
- ❑ Objects

ECMAScript is simply a description, defining all the properties, methods, and objects of a scripting language. Other languages implement ECMAScript, as JavaScript does (see Figure 1-2), as the baseline for functionality.

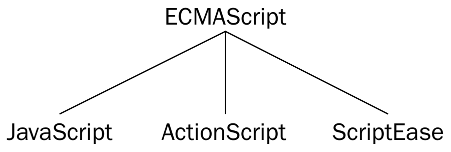

Figure 1-2

Each browser has its own implementation of the ECMAScript interface, which is then extended to contain the DOM and BOM (discussed in the following sections). There are other languages that also implement and extend ECMAScript such as Windows Scripting Host (WSH), ActionScript in Macromedia Flash and Director, and Nombas ScriptEase.

ECMAScript editions

ECMAScript is separated into editions rather than versions because it is defined in a standard called ECMA-262. Like any standard, ECMA-262 can be edited and updated. When a major update occurs, a new edition of the standard is published. The most recent edition of ECMA-262 is edition 3, released in December of 1999. The first edition of ECMA-262 was essentially the same as Netscape's JavaScript 1.1 with all browser-specific code removed, but with a few changes. First, ECMA-262 required support for the Unicode Standard (to support multiple languages). Second, it required that objects be platform-independent (Netscape's JavaScript 1.1 actually had different implementations of objects, such as the Date object, depending on the platform). This was a major reason why JavaScript 1.1 and 1.2 did not conform to the first edition of ECMA-262.

The second edition of ECMA-262 was largely editorial in nature. The standard was updated in order to get into strict agreement with ISO/IEC-16262 and didn't feature any additions, changes, or omissions. ECMAScript implementations typically don't use the second edition as a measure of conformance.

The third edition of ECMA-262 was the first real update to the standard. It provides updates to string handling, the definition of errors, and numeric outputs. It also adds support for regular expressions, new control statements, try...catch exception handling, and small changes to better prepare the standard for internationalization. To many, this marked the arrival of ECMAScript as a true programming language.

What does ECMAScript conformance mean?

In ECMA-262, the definition of ECMAScript conformance is laid out. A scripting language must subscribe to four basic tenets:

- ❑ A conforming implementation must support all "types, values, objects, properties, functions, and program syntax and semantics" (ECMA-262, p. 1) as they are described in ECMA-262.
- ❑ A conforming implementation must support the Unicode Character Standard.
- ❑ A conforming implementation may add "additional types, values, objects, properties, and functions" that are not specified in ECMA-262. ECMA-262 describes these additions as primarily new objects or new properties of objects not given in the specification.
- ❑ A conforming implementation may support "program and regular expression syntax" that are not defined in ECMA-262 (meaning that the built-in regular expression support is allowed to be altered and extended).

All implementations of ECMAScript must be in agreement with these criteria.

ECMAScript support in Web browsers

Netscape Navigator 3.0 shipped with JavaScript 1.1 in 1996. That same JavaScript 1.1 specification was then submitted to the ECMA as a proposal for a new standard. With JavaScript's explosive popularity, Netscape was very happy to start developing version 1.2. One problem: ECMA hadn't yet accepted Netscape's proposal.

A little after Netscape Navigator 3.0 was released, Microsoft introduced Internet Explorer 3.0. This version of IE shipped with JScript 1.0 (Microsoft's name for its JavaScript implementation), which was supposed to be equivalent to JavaScript 1.1. However, because of undocumented and improperly replicated features, JScript 1.0 fell far short of JavaScript 1.1.

Netscape Navigator 4.0 was shipped in 1997 with JavaScript 1.2 before the first edition of ECMA-262 was finalized; ECMA-262 was accepted and standardized later that year. As a result, JavaScript 1.2 is not compliant to the first edition of ECMAScript, even though ECMAScript was supposed to be based on JavaScript 1.1.

The next update to JScript occurred in Internet Explorer 4.0 with version JScript 3.0 (version 2.0 was released in Microsoft's Internet Information Server version 3.0 but was never included in a browser). Microsoft put out a press release touting JScript 3.0 as the first truly ECMA-compliant scripting language in the world. At that time, ECMA-262 hadn't yet been finalized, so JScript 3.0 suffered the same fate as JavaScript 1.2: It did not comply with the final ECMAScript standard.

Netscape opted to update its JavaScript implementation in Netscape Navigator 4.06. JavaScript 1.3 brought Netscape into full compliance with ECMAScript Edition 1. Netscape added support for the Unicode standard and made all objects platform-independent while keeping the features that were introduced in JavaScript 1.2.

When Netscape released its source code to the public as the Mozilla project, it was anticipated that JavaScript 1.4 would be shipped with Netscape Navigator 5.0. However, a radical decision to completely redesign the Netscape code from the bottom up threw a monkey wrench into the works. JavaScript 1.4 was only released as a server-side language for the Netscape Enterprise Server and never made it into a Web browser.

Today, all popular Web browsers comply with the third edition of ECMA-262. The following table lists ECMAScript support in the most popular Web browsers:

Browser	ECMAScript Compliance
Netscape Navigator 2.0	–
Netscape Navigator 3.0	–
Netscape Navigator 4.0–4.05	–
Netscape Navigator 4.06–4.79	Edition 1
Netscape 6.0+ (Mozilla 0.6.0+)	Edition 3
Internet Explorer 3.0	–
Internet Explorer 4.0	–

Table continued on following page

Browser	ECMAScript Compliance
Internet Explorer 5.0	Edition 1
Internet Explorer 5.5+	Edition 3
Opera 6.0–7.1	Edition 2
Opera 7.2+	Edition 3
Safari 1.0+/Konqueror ~2.0+	Edition 3

The Document Object Model (DOM)

The *Document Object Model* (DOM) is an application programming interface (API) for HTML as well as XML. The DOM maps out an entire page as a document composed of a hierarchy of nodes. Each part of an HTML or XML page is a derivative of a node. Consider the following HTML page:

```
<html>
    <head>
        <title>Sample Page</title>
    </head>
    <body>
        <p>Hello World!</p>
    </body>
</html>
```

This code can be diagrammed into a hierarchy of nodes using the DOM (see Figure 1-3).

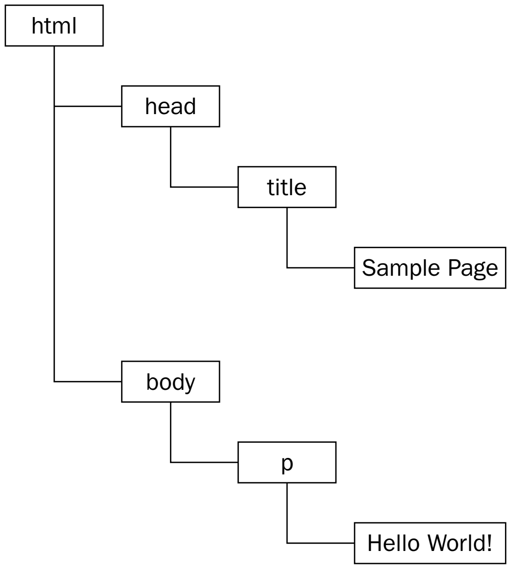

Figure 1-3

By creating a tree to represent a document, the DOM allows developers an unprecedented level of control over its content and structure. Nodes can easily be removed, added, and replaced by using the DOM API.

Why the DOM is necessary

With Internet Explorer 4.0 and Netscape Navigator 4.0 each supporting different forms of Dynamic HTML (DHTML), developers for the first time could alter the appearance and content of a Web page without reloading it. This represented a tremendous step forward in Web technology, but also a huge problem. Netscape and Microsoft each went its own way in developing DHTML, thus ending the period when Web developers could write a single HTML page that could be accessed by any Web browser.

It was decided that something had to be done to preserve the cross-platform nature of the Web. The fear was that, if someone didn't rein in Netscape and Microsoft, the Web would develop into two distinct factions that were exclusive to targeted browsers. It was then that the World Wide Web Consortium (W3C), the body charged with creating standards for Web communication, began working on the DOM.

DOM levels

DOM Level 1 became a W3C recommendation in October of 1998. It consisted of two modules: the DOM Core, which provided a way to map the structure of an XML-based document to allow for easy access to and manipulation of any part of a document, and the DOM HTML, which extended the DOM Core by adding HTML-specific objects and methods.

Note that the DOM is not JavaScript-specific, and indeed has been implemented in numerous other languages. For Web browsers, however, the DOM has been implemented using ECMAScript and now makes up a large part of the JavaScript language.

Whereas DOM Level 1's only goal was to map out the structure of a document, DOM Level 2's aims were much broader. This extension to the original DOM added support for mouse and user interface events (long supported by DHTML), ranges, traversals (methods to iterate over a DOM document), and support for Cascading Style Sheets (CSS) through object interfaces. The original DOM Core introduced in Level 1 was also extended to include support for XML namespaces.

DOM Level 2 introduced several new modules of the DOM to deal with new types of interfaces:

❑ DOM Views — describes interfaces to keep track of the various views of a document (that is, the document before CSS styling and the document after CSS styling)

❑ DOM Events — describes interfaces for events

❑ DOM Style — describes interfaces to deal with CSS-based styles

❑ DOM Traversal and Range — describes interfaces to traverse and manipulate a document tree

DOM Level 3 further extends the DOM with the introduction of methods to load and save documents in a uniform way (contained in a new module called DOM Load and Save) as well as methods to validate a document (DOM Validation). In Level 3, the DOM Core is extended to support all of XML 1.0, including XML Infoset, XPath, and XML Base.

When reading about the DOM, you may come across references to DOM Level 0. Note that there is no standard called DOM Level 0; it is simply a reference point in the history of the DOM (DOM Level 0 is considered to be the original DHTML supported in Internet Explorer 4.0 and Netscape Navigator 4.0).

Other DOMs

Aside from the DOM Core and DOM HTML interfaces, several other languages have had their own DOM standards published. The languages are XML-based and each DOM adds methods and interfaces unique to that language:

❑ Scalable Vector Graphics (SVG) 1.0

❑ Mathematical Markup Language (MathML) 1.0

❑ Synchronized Multimedia Integration Language (SMIL)

Additionally, other languages have developed their own DOM implementations, such as Mozilla's XML User Interface Language (XUL). However, only the languages in the preceding list are standard recommendations from W3C.

DOM support in Web browsers

The DOM was already a standard for some time before Web browsers started implementing it. Internet Explorer took first stab in version 5.0, but it actually didn't have any realistic DOM support until version 5.5, when it implemented most of DOM Level 1. Internet Explorer hasn't introduced new DOM functionality since that time.

For Netscape, no DOM support existed until Netscape 6 (Mozilla 0.6.0) was introduced. To date, Mozilla has the best support for the DOM, implementing all of Level 1, nearly all of Level 2, and some parts of Level 3. (The goal of the Mozilla development team was to build a 100% standards-compliant browser, and their work paid off.)

Latecomers such as Opera, which didn't add DOM support until version 7.0, and Safari, which has implemented most of DOM Level 1, are mostly on par with Internet Explorer 5.5; and in some cases, they exceed it. However, all the browsers are still a distant second to Mozilla as far as DOM support goes. The following table shows DOM support for popular browsers:

Browser	**DOM Compliance**
Netscape Navigator 1.0–4.x	–
Netscape 6.0+ (Mozilla 0.6.0+)	Level 1, Level 2, Level 3 (partial)
Internet Explorer 2.0–4.x	–
Internet Explorer 5.0	Level 1 (minimal)
Internet Explorer 5.5+	Level 1 (almost all)
Opera 1.0–6.0	–
Opera 7.0+	Level 1 (almost all), Level 2 (partial)
Safari 1.0+/Konqueror ~2.0+	Level 1

The Browser Object Model (BOM)

The Internet Explorer 3.0 and Netscape Navigator 3.0 browsers feature a *Browser Object Model* (BOM) that allows access and manipulation of the browser window. Using the BOM, developers can move the window, change text in the status bar, and perform other actions that do not directly relate to the page content. What makes the BOM truly unique, and often problematic, is that it is the only part of a JavaScript implementation that has no related standard.

Primarily, the BOM deals with the browser window and frames, but generally any browser-specific extension to JavaScript is considered to be a part of the BOM. Such things include:

- The capability to pop up new browser windows.
- The capability to move, resize, and close browser windows.
- The navigator object, which provides detailed information about the Web browser.
- The location object, which gives detailed information about the page loaded in the browser.
- The screen object, which gives detailed information about the user's screen resolution.
- Support for cookies.
- Internet Explorer extends the BOM to include the ActiveXObject class, which can be used to instantiate ActiveX objects through JavaScript.

Because no standards exist for the BOM, each browser has its own implementation. There are some *de facto* standards, such as having a window object and a navigator object, but each browser defines its own properties and methods for these and other objects. Chapter 5, "JavaScript in the Browser," goes into more detail about the implementation differences.

Summary

This chapter introduced JavaScript as a client-side scripting language for Web browsers. You learned about the various parts that make up a complete JavaScript implementation:

- ECMAScript, the core of JavaScript, describes the language syntax and basic objects.
- The Document Object Model (DOM) describes methods and interfaces for working with the content of a Web page.
- The Browser Object Model (BOM) describes methods and interfaces for interacting with the browser.

Additionally, you explored the history of JavaScript to gain an understanding of how various parts of the language developed and how browsers historically have dealt with the implementation of standards.