CHAPTER 8

The DOM Reference

As described in Chapter 3, the DOM isn’t JavaScript and it isn’t part of the ECMA-262 standard. Instead, the DOM is specified by the W3C across multiple specifications. However, much of what you’ll be doing with JavaScript will involve the DOM, so it’s important to cover it.

We’ve already covered the important aspects of the DOM in Chapter 3, including:

- The history of the DOM and the different specifications that comprise it
- Accessing elements in the DOM
- Creating, deleting, and modifying elements in the DOM
- DOM events: handlers, custom events, etc.

This chapter provides a reference for the topics covered in Chapter 3, as well as several other common features of the DOM that we didn’t cover in Chapter 3. Because the DOM specifications are quite large, this chapter won’t be exhaustive. Instead, we’ll focus on the features that are the most commonly used. We’ll also cover features that offer highly useful functionality but perhaps aren’t commonly used, either because they’re new or because they’re not often covered in references.

Browser Support

As mentioned in Chapter 3, the DOM has varying support across browsers, and between versions of individual browsers. This reference presumes so-called “modern” browsers: Internet Explorer 9 and later, and the latest versions of auto-updating browsers such as Safari, Firefox, and Chrome. If a particular feature has support problems in these target browsers, we’ll mention it. If your project needs to target older browsers, you should make sure the features you want to use are supported in your target browsers. A good reference for ensuring this are the compatibility tables at the QuirksMode.org web site: DOM features are covered at www.quirksmode.org/dom/w3c_core.html, and DOM events are covered at www.quirksmode.org/dom/events/.

DOM Objects

The most common work you’ll be doing with the DOM will involve accessing and manipulating documents and their elements. In this reference we’ll focus on the DOM objects that are the most relevant to those tasks:

- window: The window object models the browser window itself, where the document is loaded. It includes properties and methods to handle scrolling the window, positioning the browser, etc.
- document: The document object models the document. It has properties and methods for accessing and modifying the contents of the document.
- element: The element object is an abstract object (meaning it is not something you access directly, like window or document, but rather serves as a template from which other objects inherit) that defines the properties and methods exposed on the elements contained within a document. As you work with DOM elements, all of element’s properties and methods will be available on them.

We’ll cover these objects in the preceding order (rather than alphabetically, as we did for the main JavaScript objects in the reference in Chapter 5) because it represents a progression of containers: the window object contains the document object, and the document object contains element objects.

## The Window Object Reference

The window object is the top of the DOM tree and represents a document loaded into the browser. Typically you’ll have only one document at a time loaded into the browser, but you can load more than one through the use of iframes. Since each document needs its own window object, by default the window object is an array-like object: the main object represents the main document, the indexed entries represent subdocuments loaded within iframes, and a length property represents the number of subdocuments. Thus, if you have only the main document with no subdocuments, the window.length property will be 0. Each iframe is its own window object, and if a given iframe has subdocuments within it, then it too will have indexed elements with the number of subdocuments represented in its length property as well.

Subdocuments can be accessed via their indices; they are in the same order as they appear in the document. Through the window.parent property, a script in a subdocument can access its parent document. As a result, any script in a document loaded into the browser can have access to any other document loaded into the browser. For security reasons, this access is limited by the Single Origin Policy.

---

**Note** The Single Origin Policy is a security feature in browsers that is designed to prevent malicious scripts from accessing content they shouldn’t. The policy basically says that scripts served from a particular site can access only documents served from that same site. More specifically, both documents must have been served using the same protocol (HTTP or HTTPS) and port (if one was specified), from the same host. If any of those are different, access between documents is not permitted.

---

The window object also serves as the global context for JavaScript. Each document therefore has its own global context. Because the window object is the global context, you do not need to preface any of its properties or methods with the window. identifier. For example, to access the location property, you can simply use location rather than window.location. However, some properties and methods are accessed with the window. identifier for the sake of explicitness. (For details on creating and managing your own properties and methods in the global scope, see Chapter 2.) In this section we’ll explicitly use the window. reference.

## Properties

In addition to serving as the global context for JavaScript, the window object has its own properties that you can access with your scripts. These properties represent the various aspects of the browser window and the document loaded within: the URL of the document, the geometry of the window, etc.