Pro JavaScript™ Techniques

John Resig

Apress®

Pro JavaScript™ Techniques

Copyright © 2006 by John Resig

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-727-9

ISBN-10 (pbk): 1-59059-727-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the US and other countries. Apress, Inc., is not affiliated with Sun Microsystems, Inc., and this book was written without endorsement from Sun Microsystems, Inc.

Lead Editor: Chris Mills
Technical Reviewer: Dan Webb
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Tracy Brown Collins
Copy Edit Manager: Nicole Flores
Copy Editor: Jennifer Whipple
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: April Eddy
Indexer: Broccoli Information Management
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section and on the book's web site at `http://jspro.org`.

CHAPTER 1

Modern JavaScript Programming

The evolution of JavaScript has been gradual but persistent. Over the course of the past decade, the perception of JavaScript has evolved from a simple toy language into a respected programming language used by corporations and developers across the globe to make incredible applications. The modern JavaScript programming language—as it has always been—is solid, robust, and incredibly powerful. Much of what I'll be discussing in this book will show what makes modern JavaScript applications so different from what they used to be. Many of the ideas presented in this chapter aren't new by any stretch, but their acceptance by thousands of intelligent programmers has helped to refine their use and to make them what they are today. So, without further ado, let's look at modern JavaScript programming.

Object-Oriented JavaScript

From a language perspective, there is absolutely nothing modern about object-oriented programming or object-oriented JavaScript; JavaScript was designed to be a completely object-oriented language from the start. However, as JavaScript has "evolved" in its use and acceptance, programmers of other languages (such as Ruby, Python, and Perl) have taken note and begun to bring their programmatic idioms over to JavaScript.

Object-oriented JavaScript code looks and behaves differently from other object-capable languages. I'll go into depth, discussing the various aspects of what makes it so unique, in Chapter 2, but for now, let's look at some of the basics to get a feel for how modern JavaScript code is written. An example of two object constructors can be found in Listing 1-1, demonstrating a simple object pairing that can be used for lectures in a school.

Listing 1-1. *Object-Oriented JavaScript Representing a Lecture and a Schedule of Lectures*

```
// The constructor for our 'Lecture'
// Takes two strings, name and teacher
function Lecture( name, teacher ) {
    // Save them as local properties of the object
    this.name = name;
    this.teacher = teacher;
}
```

```
// A method of the Lecture class, used to generate
// a string that can be used to display Lecture information
Lecture.prototype.display = function(){
    return this.teacher + " is teaching " + this.name;
};

// A Schedule constructor that takes in an
// array of lectures
function Schedule( lectures ) {
    this.lectures = lectures;
}

// A method for constructing a string representing
// a Schedule of Lectures
Schedule.prototype.display = function(){
    var str = "";

    // Go through each of the lectures, building up
    // a string of information
    for ( var i = 0; i < this.lectures.length; i++ )
        str += this.lectures[i].display() + " ";

    return str;
};
```

As you can probably see from the code in Listing 1-1, most of the object-oriented fundamentals are there but are structured differently from other more common object-oriented languages. You can create object constructors and methods, and access and retrieve object properties. An example of using the two classes in an application is shown in Listing 1-2.

Listing 1-2. *Providing a User with List of Classes*

```
// Create a new Schedule object and save it in
// the variable 'mySchedule'
var mySchedule = new Schedule([
    // Create an array of the Lecture objects, which
    // are passed in as the only argument to the Lecture object
    new Lecture( "Gym", "Mr. Smith" ),
    new Lecture( "Math", "Mrs. Jones" ),
    new Lecture( "English", "TBD" )
]);

// Display the Schedule information as a pop-up alert
alert( mySchedule.display() );
```

With the acceptance of JavaScript among programmers, the use of well-designed object-oriented code has also become more popular. Throughout the book I'll attempt to show different pieces of object-oriented JavaScript code that I think best exemplifies code design and implementation.

Testing Your Code

After establishing a good object-oriented code base, the second aspect of developing professional-quality JavaScript code is to make sure that you have a robust code-testing environment. The need for proper testing is especially apparent when you develop code that will be actively used or maintained by other developers. Providing a solid basis for other developers to test against is essential for maintaining code development practices.

In Chapter 4, you'll look at a number of different tools that can be used to develop a good testing/use case regime along with simple debugging of complex applications. One such tool is the Firebug plug-in for Firefox. Firebug provides a number of useful tools, such as an error console, HTTP request logging, debugging, and element inspection. Figure 1-1 shows a live screenshot of the Firebug plug-in in action, debugging a piece of code.

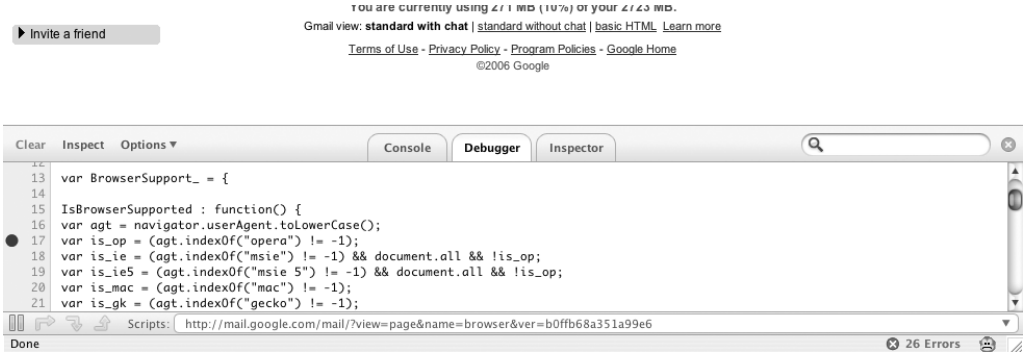

Figure 1-1. *A screenshot of the Firefox Firebug plug-in in action*

The importance of developing clean, testable code cannot be overstated. Once you begin developing some clean object-oriented code and pairing it together with a proper testing suite, I'm sure you'll be inclined to agree.

Packaging for Distribution

The final aspect of developing modern, professional JavaScript code is the process of packaging code for distribution or real-world use. As developers have started to use more and more JavaScript code in their pages, the possibility for conflicts increases. If two JavaScript libraries both have a variable named *data* or both decide to add events differently from one another, disastrous conflicts and confusing bugs can occur.

The holy grail of developing a successful JavaScript library is the ability for the developer to simply drop a <script> pointer to it and have it work with no changes. A number of techniques and solutions exist that developers use to keep their code clean and universally compatible.

The most popular technique for keeping your code from influencing or interfering with other JavaScript code is the use of namespaces. The ultimate (but not necessarily the best or

most useful) example of this in action is a public user interface library developed by Yahoo, which is available for anyone to use. An example of using the library is shown in Listing 1-3.

Listing 1-3. *Adding an Event to an Element Using the Heavily Namespaced Yahoo UI Library*

```
// Add a mouseover event listener to the element that has an
// ID of 'body'
YAHOO.util.Event.addListener('body','mouseover',function(){

    // and change the background color of the element to red
    this.style.backgroundColor = 'red';

});
```

One problem that exists with this method of namespacing, however, is that there is no inherent consistency from one library to another on how it should be used or structured. It is on this point that central code repositories such as JSAN (JavaScript Archive Network) become immensely useful. JSAN provides a consistent set of rules for libraries to be structured against, along with a way to quickly and easily import other libraries that your code relies upon. A screenshot of the main distribution center of JSAN is shown in Figure 1-2.

I will discuss the intricacies of developing clean, packageable code in Chapter 3. Additionally, the importance of other common stumbling points, such as event-handling collision, will be discussed in Chapter 6.

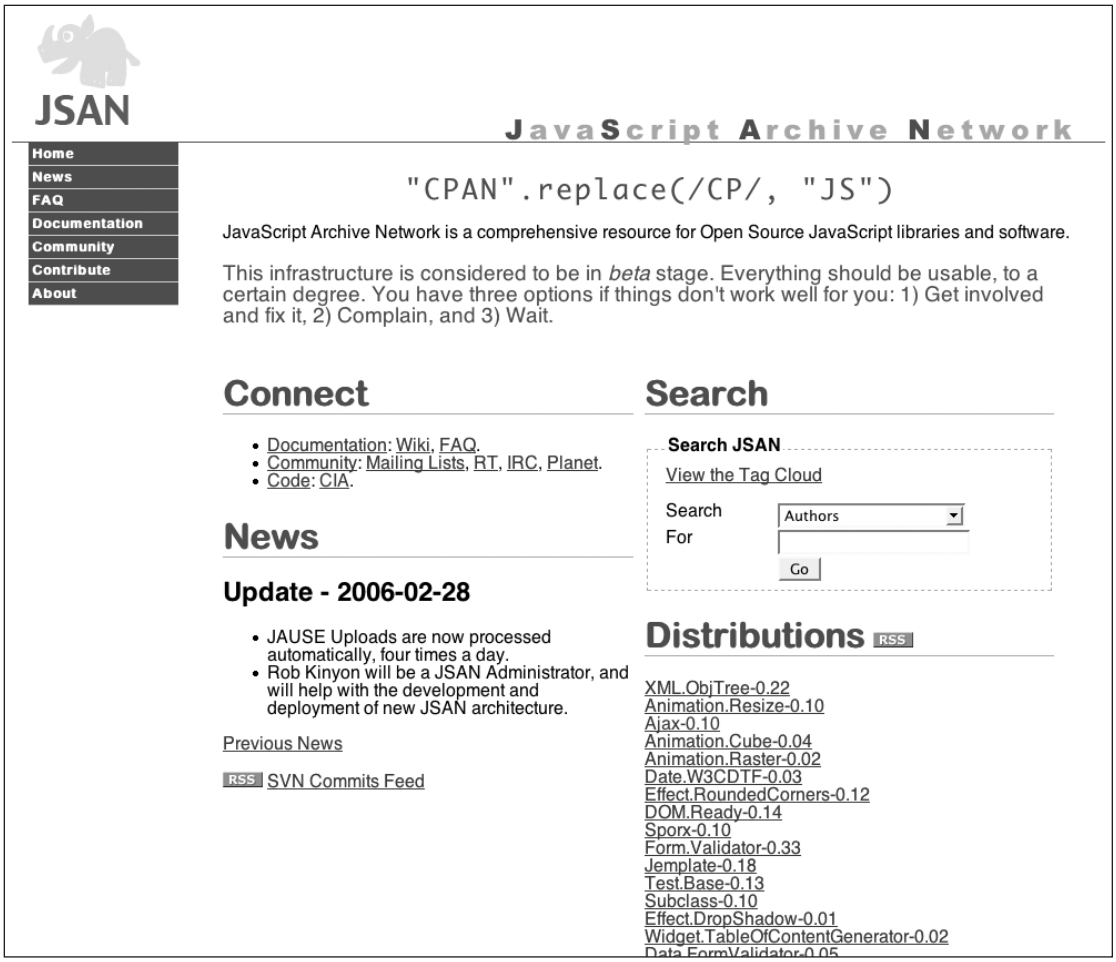

Figure 1-2. *A screenshot of the public JSAN code repository*

Unobtrusive DOM Scripting

Built upon a core of good, testable code and compliant distributions is the concept of unobtrusive DOM scripting. Writing unobtrusive code implies a complete separation of your HTML content: the data coming from the server, and the JavaScript code used to make it all dynamic. The most important side effect of achieving this complete separation is that you now have code that is perfectly downgradeable (or upgradeable) from browser to browser. You can use this to offer advanced content to browsers that support it, while still downgrading gracefully for browsers that don't.

Writing modern, unobtrusive code consists of two aspects: the Document Object Model (DOM), and JavaScript events. In this book I explain both of these aspects in depth.

The Document Object Model

The DOM is a popular way of representing XML documents. It is not necessarily the fastest, lightest, or easiest to use, but it is the most ubiquitous, with an implementation existing in most web development programming languages (such as Java, Perl, PHP, Ruby, Python, and JavaScript). The DOM was constructed to provide an intuitive way for developers to navigate an XML hierarchy.

Since valid HTML is simply a subset of XML, having an efficient way to parse and browse DOM documents is absolutely essential for making JavaScript development easier. Ultimately, the majority of interaction that occurs in JavaScript is between JavaScript and the different HTML elements contained within a web page; and the DOM is an excellent tool for making this process simpler. Some examples of using the DOM to navigate and find different elements within a page and then manipulate them can be found in Listing 1-4.

Listing 1-4. *Using the Document Object Model to Locate and Manipulate Different DOM Elements*

```
<html>
<head>
    <title>Introduction to the DOM</title>
    <script>
    // We can't manipulate the DOM until the document
    // is fully loaded
    window.onload = function(){

        // Find all the <li> elements in the document
        var li = document.getElementsByTagName("li");

        // and add a ared border around all of them
        for ( var j = 0; j < li.length; j++ ) {
            li[j].style.border = "1px solid #000";
        }

        // Locate the element with an ID of 'everywhere'
        var every = document.getElementById( "everywhere" );

        // and remove it from the document
        every.parentNode.removeChild( every );

    };
    </script>
</head>
<body>
    <h1>Introduction to the DOM</h1>
    <p class="test">There are a number of reasons why the
        DOM is awesome, here are some:</p>
```

```
    <ul>
        <li id="everywhere">It can be found everywhere.</li>
        <li class="test">It's easy to use.</li>
        <li class="test">It can help you to find what you want, really quickly.</li>
    </ul>
</body>
</html>
```

The DOM is the first step to developing unobtrusive JavaScript code. By being able to quickly and simply navigate an HTML document, all resulting JavaScript/HTML interactions become that much simpler.

Events

Events are the glue that holds together all user interaction within an application. In a nicely designed JavaScript application, you're going to have your data source and its visual representation (inside of the HTML DOM). In order to synchronize these two aspects, you're going to have to look for user interactions and attempt to update the user interface accordingly. The combination of using the DOM and JavaScript events is the fundamental union that makes all modern web applications what they are.

All modern browsers provide a number of events that are fired whenever certain interactions occur, such as the user moving the mouse, striking the keyboard, or exiting the page. Using these events, you can register code that will be executed whenever the event occurs. An example of this interaction is shown in Listing 1-5, where the background color of the s change whenever the user moves his mouse over them.

Listing 1-5. *Using the DOM and Events to Provide Some Visual Effects*

```
<html>
<head>
    <title>Introduction to the DOM</title>
    <script>
    // We can't manipulate the DOM until the document
    // is fully loaded
    window.onload = function(){

        // Find all the <li> elements, to attach the event handlers to them
        var li = document.getElementsByTagName("li");
        for ( var i = 0; i < li.length; i++ ) {

            // Attach a mouseover event handler to the <li> element,
            // which changes the <li>s background to blue.
            li[i].onmouseover = function() {
                this.style.backgroundColor = 'blue';
            };
```

```
            // Attach a mouseout event handler to the <li> element
            // which changes the <li>s background back to its default white
            li[i].onmouseout = function() {
                this.style.backgroundColor = 'white';
            };

        }

    };
    </script>
</head>
<body>
    <h1>Introduction to the DOM</h1>
    <p class="test">There are a number of reasons why the
        DOM is awesome, here are some:</p>
    <ul>
        <li id="everywhere">It can be found everywhere.</li>
        <li class="test">It's easy to use.</li>
        <li class="test">It can help you to find what you want, really quickly.</li>
    </ul>
</body>
</html>
```

JavaScript events are complex and diverse. Much of the code and applications in this book utilize events in one way or another. Chapter 6 and Appendix B are completely dedicated to events and their interactions.

JavaScript and CSS

Building upon your base of DOM and event interactions comes dynamic HTML. At its core, dynamic HTML represents the interactions that occur between JavaScript and the CSS information attached to DOM elements.

Cascading style sheets (CSS) serve as the standard for laying out simple, unobtrusive web pages that still afford you (the developer) the greatest amount of power while providing your users with the least amount of compatibility issues. Ultimately, dynamic HTML is about exploring what can be achieved when JavaScript and CSS interact with each other and how you can best use that combination to create impressive results.

For some examples of advanced interactions, such as drag-and-drop elements and animations, take a look at Chapter 7, where they are discussed in depth.

Ajax

Ajax, or Asynchronous JavaScript and XML, is a term coined in the article "Ajax: A New Approach to Web Applications" (`http://www.adaptivepath.com/publications/essays/archives/000385.php`) by Jesse James Garrett, cofounder and president of Adaptive Path, an information architecture firm. It describes the advanced interactions that occur between the client and the server, when requesting and submitting additional information.

The term *Ajax* encompasses hundreds of permutations for data communication, but all center around a central premise: that additional requests are made from the client to the server even after the page has completely loaded. This allows application developers to create additional interactions that can involve the user beyond the slow, traditional flow of an application. Figure 1-3 is a diagram from Garrett's Ajax article that shows how the flow of interaction within an application changes due to the additional requests that are made in the background (and most likely without the user's knowledge).

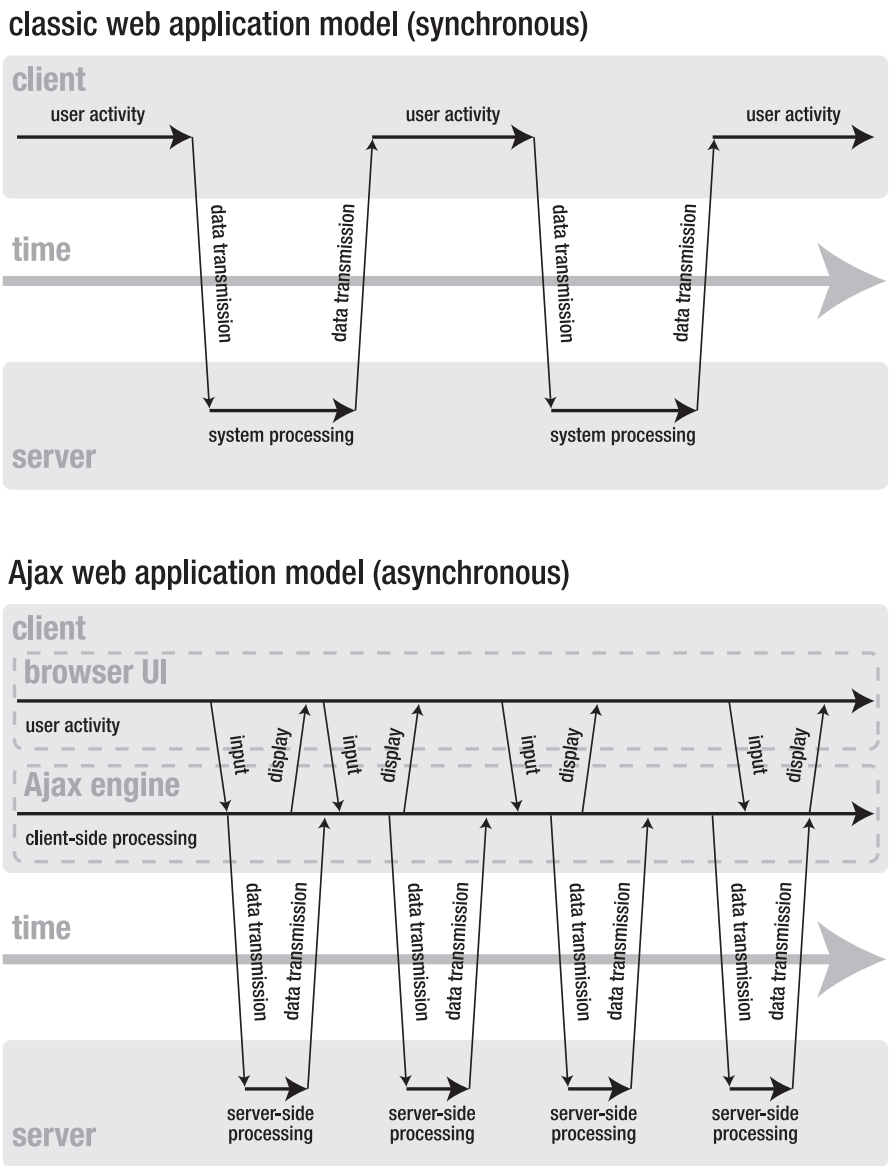

Figure 1-3. *A diagram from the article "Ajax: A New Approach to Web Applications," showing the advanced, asynchronous interaction that occurs between the client and a server*

Since the original release of the Garrett article, the interest of users, developers, designers, and managers has been piqued, allowing for an explosion of new applications that make use of this advanced level of interaction. Ironically, while there has been this resurgence in interest, the technology behind Ajax is rather old (having been used commercially since around the year 2000). The primary difference, however, is that the older applications utilized browser-specific means of communicating with the server (such as Internet Explorer–only features). Since all modern browsers support XMLHttpRequest (the primary method for sending or receiving XML data from a server), the playing field has been leveled, allowing for everyone to enjoy its benefits.

If one company has been at the forefront of making cool applications using Ajax technology, it's Google. One highly interactive demo that it released just before the original Ajax article came out is Google Suggest. The demo allows you to type your query and have it be autocompleted in real time; this is a feature that could never be achieved using old page reloads. A screenshot of Google Suggest in action is shown in Figure 1-4.

Figure 1-4. *A screenshot of Google Suggest, an application available at the time of Garrett's Ajax article that utilized the asynchronous XML techniques*

Additionally, another revolutionary application of Google is Google Maps, which allows the user to move around a map and see relevant, local results displayed in real time. The level of speed and usability that this application provides by using Ajax techniques is unlike any other mapping application available and has completely revolutionized the online mapping market as a result. A screenshot of Google Maps is shown in Figure 1-5.

Even though very little has physically changed within the JavaScript language, during the past couple years, the acceptance of JavaScript as a full-blown programming environment by such companies as Google and Yahoo shows just how much has changed in regard to its perception and popularity.

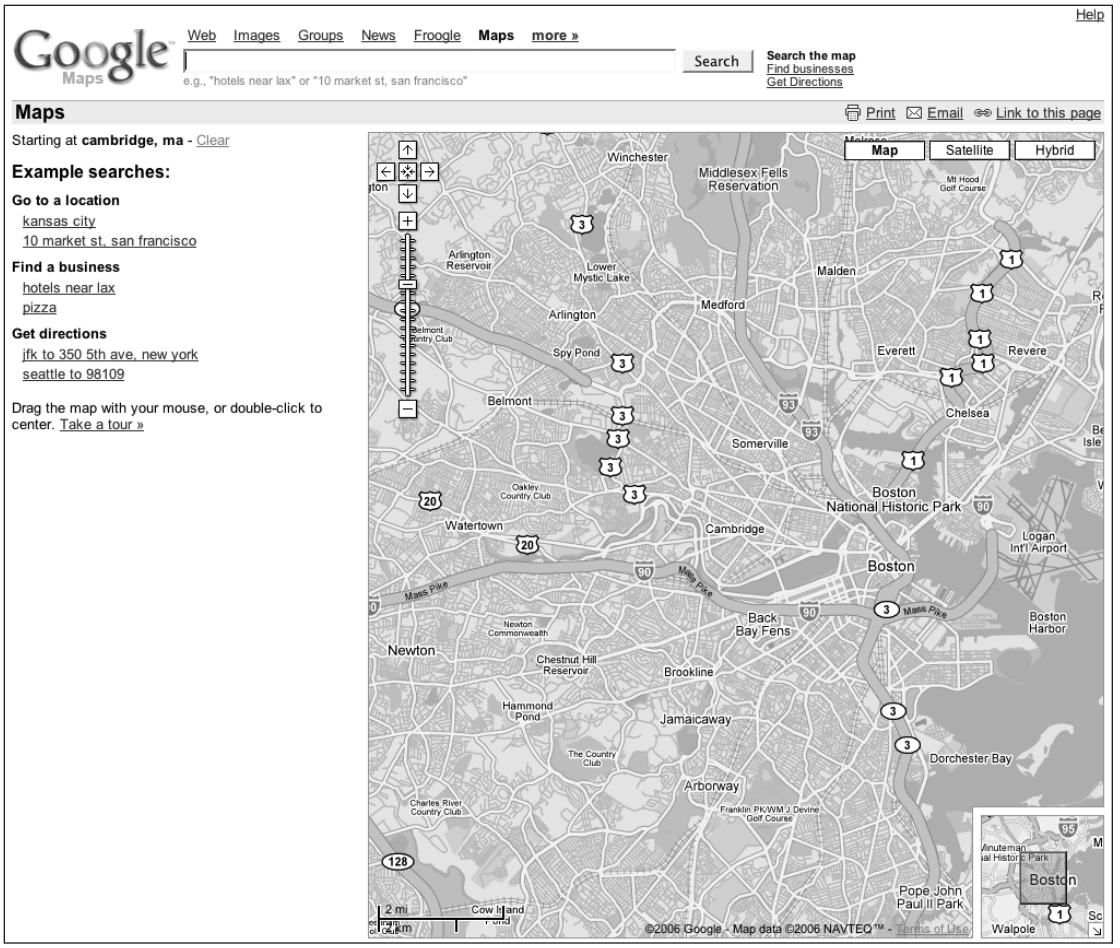

Figure 1-5. *Google Maps, which utilizes a number of Ajax techniques to dynamically load location information*

Browser Support

The sad truth of JavaScript development is that since it is so tied to the browsers that implement and support it, it is also at the mercy of whichever browsers are currently the most popular. Since users don't necessarily use the browsers with the best JavaScript support, we're forced to pick and choose which features are most important.

What many developers have begun to do is cut off support for browsers that simply cause too much trouble when developing for them. It's a delicate balance between supporting browsers due to the size of their user base and supporting them because they have a feature that you like.

Recently Yahoo released a JavaScript library that can be used to extend your web applications. Along with the library, it also released some design pattern guidelines for web developers to follow. The most important document to come out of it (in my opinion) is Yahoo's

official list of browsers that it does and doesn't support. While anyone, and any corporation, can do something similar, having a document provided by one of the most trafficked web sites on the Internet is entirely invaluable.

Yahoo developed a graded browser support strategy that assigns a certain grade to a browser and provides different content to it based upon its capabilities. Yahoo gives browsers three grades: A, X, and C:

- A-grade browsers are fully supported and tested, and all Yahoo applications are guaranteed to work in them.
- An X-grade browser is an A-grade browser that Yahoo knows exists but simply does not have the capacity to test thoroughly, or is a brand-new browser that it's never encountered before. X-grade browsers are served with the same content as A-grade browsers, in hopes that they'll be able to handle the advanced content.
- C-grade browsers are known as "bad" browsers that do not support the features necessary to run Yahoo applications. These browsers are served the functional application contents without JavaScript, as Yahoo applications are fully unobtrusive (in that they will continue to work without the presence of JavaScript).

Incidentally, Yahoo's browser grade choices just so happen to coincide with my own, which makes it particularly appealing. Within this book, I use the term *modern browser* a lot; when I use that phrase, I mean any browser that has grade-A support deemed by the Yahoo browser chart. By giving you a consistent set of features with which to work, the learning and development experience will become much more interesting and much less painful (all by avoiding browser incompatibilities).

I highly recommend that you read through graded browser support documents (which can be found at `http://developer.yahoo.com/yui/articles/gbs/gbs.html`), including the browser support chart shown in Figure 1-6, to get a feel for what Yahoo is attempting to accomplish. By making this information available to the general web-developing public, Yahoo is providing an invaluable "gold standard" for all others to reach, which is a great thing to have.

For more information about all the browsers that are supported, see Appendix C of this book where the shortcomings and advantages of each browser are discussed in depth. More often than not, you'll find all of the A-grade browsers to be on the cutting edge of development, providing more than enough features for you to develop with.

When choosing what browsers you wish to support, the end result ultimately boils down to a set of features that your application is able to support. If you wish to support Netscape Navigator 4 or Internet Explorer 5 (for example), it would severely limit the number of features that you could use in your application, due to their lack of support for modern programming techniques.

	Win 98	Win 2000	Win XP	Mac 10.0	Mac 10.2	Mac 10.3	Mac 10.3.x	Mac 10.4
IE 7.0	n/a	n/a	A-grade	n/a	n/a	n/a	n/a	n/a
IE 6.0	A-grade	A-grade	A-grade	n/a	n/a	n/a	n/a	n/a
IE 5.5	A-grade	A-grade	n/a	n/a	n/a	n/a	n/a	n/a
IE 5.0	C-grade	C-grade	n/a	C-grade	C-grade	C-grade	C-grade	C-grade
Netscape 8.0	X-grade	X-grade	A-grade	n/a	n/a	n/a	n/a	n/a
Firefox 1.5	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade
Firefox 1.0.7	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade
Mozilla 1.7.12	X-grade	X-grade	A-grade	X-grade	X-grade	X-grade	X-grade	X-grade
Opera 8.5	X-grade	X-grade	A-grade	C-grade	C-grade	C-grade	X-grade	X-grade
Safari 1.0	n/a	n/a	n/a	X-grade	n/a	n/a	n/a	n/a
Safari 1.1	n/a	n/a	n/a	X-grade	X-grade	n/a	n/a	n/a
Safari 1.2	n/a	n/a	n/a	X-grade	X-grade	X-grade	n/a	n/a
Safari 1.3	n/a	n/a	n/a	n/a	n/a	X-grade	A-grade	n/a
Safari 2.0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	A-grade

Figure 1-6. *The graded browser support chart provided by Yahoo*

However, knowing which browsers are modern allows you to utilize the powerful features that are available in them, giving you a consistent base from which you can do further development. This consistent development base can be defined by the following set of features:

Core JavaScript 1.5: The most current, accepted version of JavaScript. It has all the features necessary to support fully functional object-oriented JavaScript. Internet Explorer 5.0 doesn't support full 1.5, which is the primary reason developers don't like to support it.

XML Document Object Model (DOM) 2: The standard for traversing HTML and XML documents. This is absolutely essential for writing fast applications.

XMLHttpRequest: The backbone of Ajax—a simple layer for initiating remote HTTP requests. All browsers support this object by default, except for Internet Explorer 5.5–6.0; however, they each support initiating a comparable object using ActiveX.

CSS: An essential requirement for designing web pages. This may seem like an odd requirement, but having CSS is essential for web application developers. Since every modern browser supports CSS, it generally boils down to discrepancies in presentation that cause the most problems. This is the primary reason Internet Explorer for Mac is less frequently supported.

The combination of all these browser features is what makes up the backbone of developing JavaScript web applications. Since all modern browsers support the previously listed features (in one way or another), it gives you a solid platform to build off of for the rest of this book. Everything discussed in this book will be based on the assumption that the browser you're using supports these features, at the very least.

Summary

This book is an attempt to completely encompass all modern, professional JavaScript programming techniques as they are used by everyone from individual developers to large corporations, making their code more usable, understandable, and interactive.

In this chapter we looked at a brief overview of everything that we're going to discuss in this book. This includes the foundations of professional JavaScript programming: writing object-oriented code, testing your code, and packaging it for distribution. Next you saw the fundamental aspects of unobtrusive DOM scripting, including a brief overview of the Document Object Model, events, and the interaction between JavaScript and CSS. Finally you looked at the premise behind Ajax and the support of JavaScript in modern browsers. All together, these topics are more than enough to take you to the level of a professional JavaScript programmer.