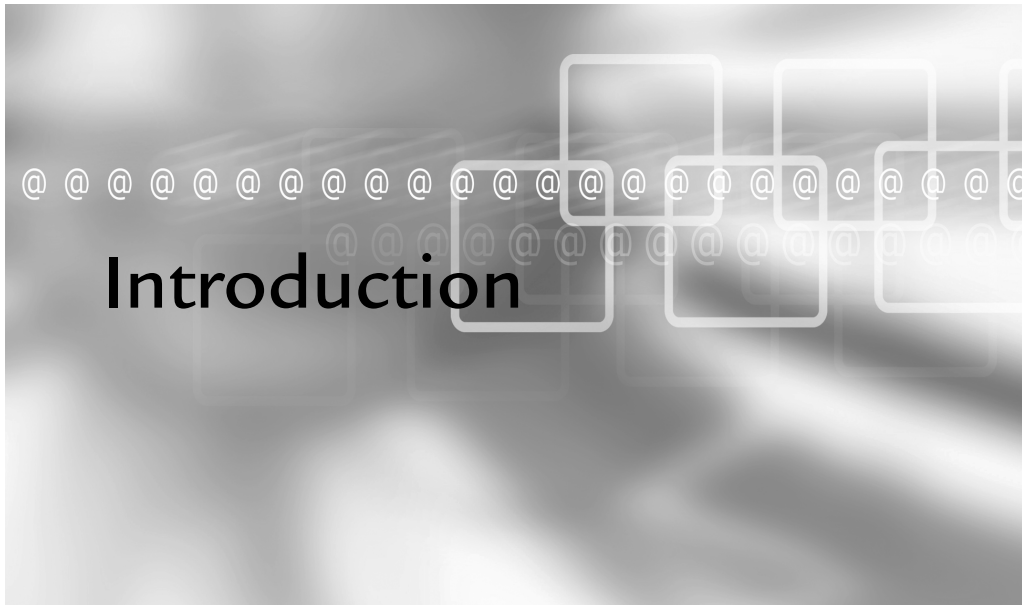

Introduction

In This Chapter

As the centerpiece of rich web application development, Ajax brings web interfaces using XHTML and CSS up to desktop application interface standards without the interfaces having to rely on plugins such as Flash or Java. Prior to JavaScript-based server interactions, interfaces had to rely solely on full-page loading, regardless of how one might have hacked a page into appearing otherwise.

Until Ajax development came along (which, incidentally, started in implementation many years before the coining of the term itself), client-side development also had no thread support. Threading, in a nutshell, allows the spawning of new lines of logic, completely independent of those before, adjacent to, or after it. C, Java, Perl, and many other languages have had this support for many years (in some cases) before client-side scripting came along in any fashionable sense. The closest JavaScript had to offer came in the form of the `setTimeout` and `setInterval` library functions, which required delayed, seemingly parallel execution rather than the actual spawning of processes. While Ajax still does not provide true threading, it does bring JavaScript one step closer.

0.1 Ajax, the Acronym

The words *Asynchronous Javascript And XML* make the acronym *Ajax*. In order to fully understand Ajax in meaning and implementation, you must understand each of its components. Even when using synchronous requests, or using JSON or some other transportation method, knowing the core aspects of Ajax can only help development practices.

Since the initial boom in popularity and resulting hype surrounding Ajax, it can get quite easy to forget what Ajax actually means and what it doesn't. Ajax does exist as an incredibly useful method of communicating with the server directly from JavaScript. It does not mean anything more than that, even if its usage can open up development methods previously unexplored in web application development.

0.1.1 Asynchronous

When requests get submitted to the server, they have no direct impact on any other simultaneous or subsequential requests. In other words, just because a request gets submitted before another request does not in any way ensure that it will receive its response from the server first. Despite the seemingly simplistic concept, asynchronistic behavior in applications often gets ignored, because asynchronicity introduces an entirely new level of complexity to client-side development.

Many Ajax-based web applications use the asynchronous flag of the `XMLHttpRequest` object solely to handle network errors (sometime without even intending to do so) rather than to keep functionality enabled during a given request. While the direct JavaScript-to-server communication provided by the `XMLHttpRequest` forms the core of the technology, the asynchronous behavior it also can provide often plays the part of the unsung hero, as it brings a wealth of flexibility and strength to client-side web applications.

0.1.2 JavaScript

JavaScript (based on ECMAScript,[1] though possibly vice-versa depending on whom you ask) has many implementations, not only in various web browsers, but also in game development and other applications needing an easy-to-learn scripting language. This book focuses on the implementation of JavaScript in various web browsers. These impleMentations of JavaScript have a wide variety of incompatibilities, from Mozilla's SpiderMonkey[2] to Safari's WebKit to Jscript and more.

Those used to server-side development or OOP (Object-Oriented Programming) may initially get thrown off by JavaScript's prototype-based object model. This, in a very basic sense, means that functions and methods called within a certain object get called *in the context* of that object. This happens because rather than an instance having

[1] Ecma International, an industry association devoted to standardizing "Information and Communication Technology (ICT) and Consumer Electronics (CE)" (What is Ecma International, www.ecma-international.org/memento/index.html), maintains the ECMA-262 standard (www.ecma-international.org/publications/standards/Ecma-262.html) which defines the scripting language of ECMAScript.

[2] http://developer.mozilla.org/en/docs/SpiderMonkey—The Gecko rendering engine's JavaScript engine written in C is used by Mozilla-based browsers such as Firefox (www.mozilla.com/products/firefox), SeaMonkey (www.mozilla.org/projects/seamonkey), Camino (www.caminobrowser.org), and Epiphany (www.gnome.org/projects/epiphany).

an explicit tie to its definition, its prototype merely lays out the basis for its structure and characteristics.

The JavaScript object, `XMLHttpRequest` (originally an ActiveX control created by Microsoft), provides the key to the entire technology conglomeration now referred to as Ajax. It provides an interface by which JavaScript can send and receive data to and from the server without requiring a full page load. Other methods exist for sending and receiving data, but they each use aspects of HTML and XHTML in ways other than designed, and, as such (while still useful in certain circumstances), they exist only as hacks.

0.1.3 XML

XML stands for *eXtensible Markup Language*, as defined by the World Wide Web Consortium (W3C; http://w3.org), and provides a very flexible, generic text format. If that seems to be a rather broad description, it should be. XML now uses spanning data storage, communication, definition, description, and presentation. In Ajax, XML refers to data transportation. The `XMLHttpRequest` object provides another useful bit of functionality along with its HTTP methods: When the server returns XML, the `XMLHttpRequest` object provides the `responseXML` attribute, which is a read-only XML document of the response.

Using XML, a very simple response from the server, with two named variables (`var1` and `var2`) each set to string values (`"first value"` and `"second value,"` respectively), might look like the following:

```
<?xml version="1.0"?>
<response>
    <var1>first value</var1>
    <var2>second value</var2>
</response>
```

Many Ajax-driven web applications use other formats of transporting data to and from the server, including:

- **URL-encoded**—Where data takes the form used by HTTP POST requests, as during a form submission such as `var1=first%20value&var2=second%20value`.
- **Raw text**—Usually for very simple data, or when responses return the exact markup for the JavaScript to insert into the current document:

```
<input type="text" name="var1" value="first value" />
<input type="text" name="var2" value="second value" />
```

- **JavaScript Object Notation (JSON)**—An increasingly popular format, JSON formats data into a subset of raw JavaScript. This not only has the advantage of instant parsing by client-side code, but also it tends to take up less bandwidth than more verbose, globally understood formats such as XML. In addition, it does so without losing the data structure as URL-encoded value pairs do:

```
{
        var1:"first value",
        var2:"second value"
}
```

0.2 This Book's Intentions

Now that the technology has progressed into general usage, the Ajax developer community has a need for books covering architecture, tuning, alternative uses of Ajax, and more. Many books and tutorials have provided good introductions, and they can show you several different ways of implementing find-as-you-type, chat widgets, and RSS/ATOM feed readers. Many of the resources out there explain, in great detail, the history of Ajax and its multiple incarnations before today's and the implementation centered on the `XMLHttpRequest` JavaScript object. See Appendix A, "Resources," at the end of this book for some choice suggestions.

This book, instead, looks at using Ajax to create rich, browser-based interfaces for enterprise-level web applications, taking into account the flexibility, reusability, scalability, and maintainability necessary for such an undertaking. Ajax does not exist in this book as the latest and greatest acronym to hit web development. It instead exists as a tool like any other—extremely useful in some instances and totally wrong in others.

For example, many reference sites would find themselves hard-pressed to use Ajax for anything of particular value to their users. Manuals and other reference materials that have large blocks of text for the user to read might come up with an Ajax reader, allowing a single, scrollable pane that late-loads content as the user scrolls though it. This sounds cool, but it destroys the ability to search the page for a particular word or phrase. It also removes the ability to read something once you've lost your Internet connection. Some reference sites add auto-suggestions to their search fields, but those tend to react too slowly for general usage unless you pre-load the entire dictionary into

the browser's memory, potentially wasting a great deal of bandwidth for a feature that only a few people might enjoy having at their disposal.

craigslist.org (see Figure 0.1) is a good example of a site that flourishes without a flashy or cluttered interface, and it has grown to provide largely free classified services and forums to 450 cities in 50 countries without so much as a single image on their main page, let alone rich application functionality. The site instead focuses on content and searching that content.

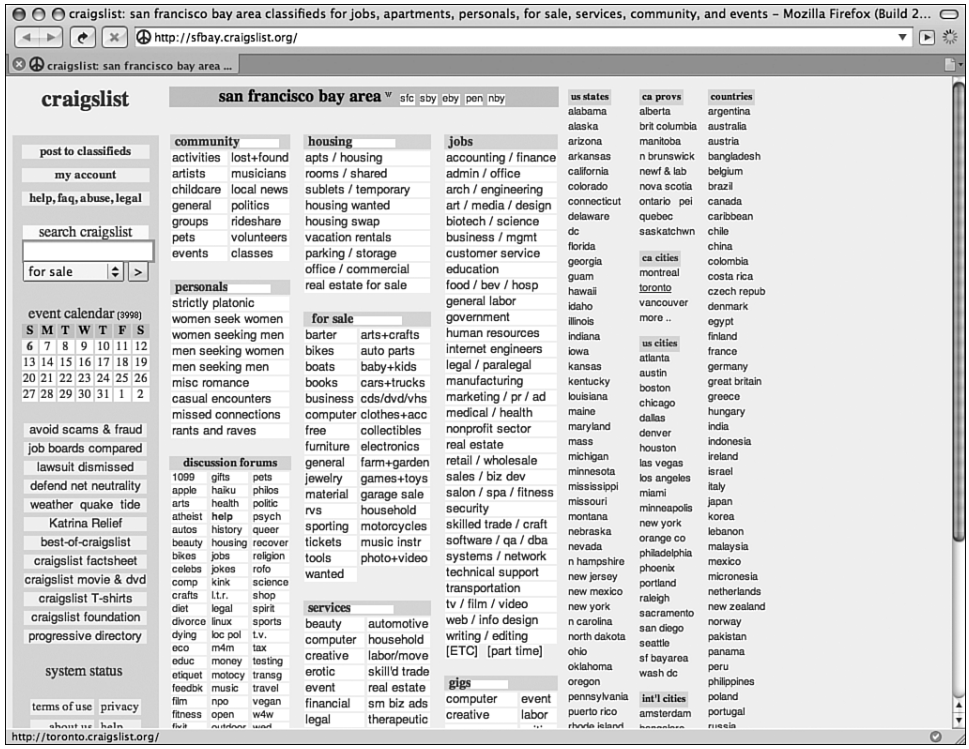

FIGURE 0.1 The default craigslist.org page.

By contrast, sites and web applications dealing with rapid browsing and editing of a large number of smaller items, or a large number of small, editable chunks of large items, flourish with Ajax usage. Google Maps (see Figure 0.2) brought everybody's attention to Ajax when it went public beta, and it uses Ajax to bring in a large number of images and metadata in chunks according to the user's interactions with the map. Web applications having a large number of transactions for a given set of elements, online games for example, save a lot of time and bandwidth by reusing the same interface multiple times to submit and display similar data.

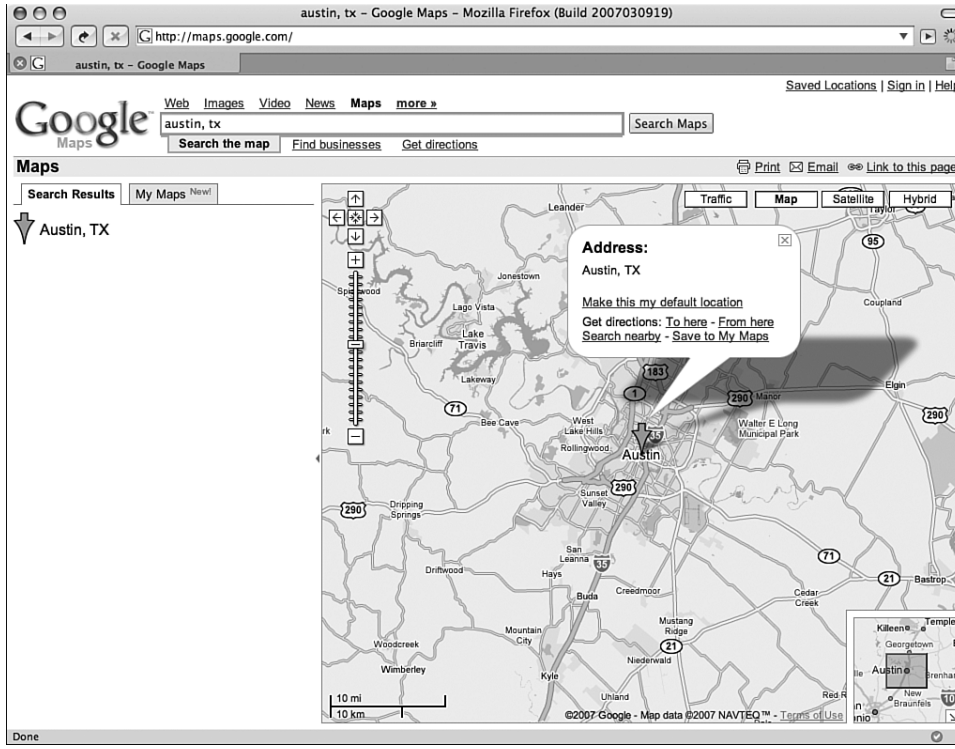

FIGURE 0.2 Google Maps focusing on Austin, TX.

No matter what your project, you should know the options for reaching your goals, which options work the best, and why. Ajax has a lot of buzz around it, both positive and negative; what it really needs, instead, is a good, solid foundation for serious, real-world application development. The OpenAjax Alliance[3] has started moving in this direction, building tools to prevent name collisions between Ajax toolkits and bringing companies and individuals together in an effort to promote stability, security, and interoperability between professional-grade toolkits.

This book covers the range of topics necessary to create a well-rounded application, regardless of the tools and technologies used. Many developers have created their own toolkits in order to abstract the actual Ajax communication layers and to speed development. Though none of the material here targets any particular toolkit, you easily could use many of those in development while still following each of the chapters.

[3] "The OpenAjax Alliance is an organization of leading vendors, open source projects, and companies using Ajax that are dedicated to the successful adoption of open and interoperable Ajax-based Web technologies. The prime objective is to accelerate customer success with Ajax by promoting a customer's ability to mix and match solutions from Ajax technology providers and by helping to drive the future of the Ajax ecosystem" (www.openajax.org).

0.3 Prerequisites for This Book

Other Ajax books have spent so much time introducing the reader to all of the technologies involved (Apache, MySQL, PHP, XHTML, JavaScript, and of course the `XMLHttpRequest` object itself) that they have not had the opportunity to delve into more advanced topics and practices. This book takes advantage of what already has been written to assume a certain level of understanding, in order to examine and explore in detail the more intricate methods of designing a web application to use Ajax. Instead of looking at some of the available AJAX frameworks, this book takes a brief look at the more experimental uses, such as game development.

As such, if you have not already worked with Ajax or some form of server-side scripting language, database, or web server, you should probably read a book like *Understanding Ajax* (Eichorn, 2006), following along with the examples. While this Introduction establishes the technologies used and referenced later in the book, it does so only as a quick overview, just as a professor provides a quick overview during the first week of a semester to refresh your memory of last semester's course.

The example code in this book uses the following technologies for each application layer. You should have a general understanding of all of these before you begin reading this book:

- **Webserver**—Apache's HTTPD (http://httpd.apache.org) version 2.0. As of this writing, the Apache foundation has released the 2.2.* branch as the primary stable branch. The example configuration directives in the book should carry over to the newer version without much deviation.
- **Database Server**—MySQL Database Server 5.0 (http://dev.mysql.com/downloads/mysql/5.0.html). The 5.0.* branch introduces a wealth of useful functionality and stability over previous versions, including stored procedures, triggers, views, and strict mode. As of this writing, MySQL AB has released the 5.1 branch as a beta.
- **Server-Side Scripting**—PHP 5.2 (www.php.net/releases/5_2_0.php). PHP 5.2 brings an input filtering extension, a JSON library enabled by default, greater ability to track file upload progress, vastly improved time zone handling, and more. While PHP 6 brings global Unicode support to PHP,[4] along with

[4] PHP does not technically pay attention to the bytes of strings. It just regards them as a numbered list of bytes. While this has the benefit of passing UTF-8 strings through PHP (even without the Multi-byte String library) unharmed, side effects can show themselves in the strangest, often most devastating, places in your application.

cleaned-up functionality, closer integration of the new PDO database extensions, even more drastic improvements to the object model, and, for some reason, `goto` (in the form of named `break` statements), the PHP group has made it available only from source so far. It has much development left on it, but should see greater adoption rates than PHP5 has seen so far.

- **Markup**—XHTML 1.1 (www.w3.org/TR/xhtml11). While XHTML 2.0 has reached its eighth public working draft, XHTML 1.1 maintains HTML compatibility while strictly enforcing XML, modules, and the progression to XHTML 2.0. Unfortunately, Internet Explorer does not really support XHTML; rather, it renders it as HTML. This does make quite a difference and holds many developers back from fully embracing the XHTML modules available to them. As such, the markup directly rendered in the browser will have `Content-type: text/html` rather than `application/xhtml+xml`, as recommended by the W3C. Technically, the specification (www.w3.org/TR/xhtml-media-types) *strongly recommends* against using `text/html` with anything beyond HTML 4 or XHTML 1.0 (HTML compatible). However, it does not forbid it, as it does with the practice of using anything aside from `text/html` with HTML 4.
- **Style**—CSS 2.1 (Cascading Style Sheets, level 2 revision 1, www.w3.org/TR/CSS21). CSS 3 introduces much of the styling and layout abilities asked for years ago and eagerly awaited by web designers; however, it has not reached a stable enough point for many of the browsers to support any more than some of the basics.[5] Even with the much-anticipated release of Internet Explorer 7 (hereafter referred to as IE or IE7), IE still fails to completely support even the CSS 2.0 specification. The IE development team worked very hard to improve the state of IE's CSS support and, while they did a fantastic job, they didn't quite make it all the way there. Because many resources (http://css-discuss.incutio.com, http://blogs.msdn.com/ie, and many more) exist to cover the hacks and fixes necessary to force IE6 and IE7 to follow your design, this book will not go into detail of how to achieve complete, pixel-perfect, cross-browser designs.
- **Client-Side Scripting**—This book will use JavaScript 1.5, together with the `XMLHttpRequest` object, which currently exists only as an informally agreed

[5] Rounded borders, multiple background images, column layout, text shadows, and transparency have all made it into the Webkit project. As of this writing, the Mozilla Gecko engine and Opera's rendering engine both have implemented most of these.

upon object and the very beginnings of a specification (www.w3.org/TR/XMLHttpRequest as part of the Web API Working Group's activities). Many Ajax-type web applications and sites use Adobe Flash for text and XML communication with the server; however, Flash development gets too specific for coverage in this book. Many of the same principles and much of the architecture covered still apply, but the implementation differs. ActionScript, also an ECMAScript implementation, actually shares the syntax, object model, and often even its development tools with JavaScript, so while the `XMLHttpRequest` object does not exist in ActionScript, and the working DOM differs, much of the other sample code should look very familiar and easy to follow.

Familiarity, at least to the point of understanding enough to port the code into your language of choice, will definitely help, though this book aims to provide the methodologies, architectures, and patterns that you can implement in your own rich web application, no matter what technology you use to drive it. The technologies listed previously have several benefits. The organizations behind them have made them freely available for download and use on a wide range of platforms and have tested them in a wide range of browsers. In addition, the technologies have large user bases and online communities ready and willing to assist you if you run into any problems.