Pro Ajax and Java™ Frameworks

Nathaniel T. Schutta and
Ryan Asleson

Apress®

Pro Ajax and Java^{TM} Frameworks

Copyright © 2006 by Nathaniel T. Schutta and Ryan Asleson

ISBN-13 (pbk): 978-1-59059-677-7

ISBN-10 (pbk): 1-59059-677-3

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

Lead Editor: Chris Mills
Technical Reviewer: John R. Fallows
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: Richard Dal Porto
Copy Edit Manager: Nicole LeClerc
Copy Editor: Hastings Hart
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Susan Glinert
Proofreader: April Eddy
Indexer: Lucie Haskins
Artist: Susan Glinert
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 source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 1

What Is Ajax?

The Internet as we know it is a very different beast than when it began. Although many find it second nature to buy the latest DVD or plan their next vacation online, in the beginning there was only text; today we have video podcasts and can easily share photos of the cats with anyone in the world. What began as a mechanism to enable greater collaboration in the research community has morphed into not only a significant sales channel but also one of the primary delivery mechanisms for modern applications.

The Rise of the Web Application

In the early days, web pages were simple static text, which worked fine for posting your latest paper or a class schedule. However, it didn't take long for people to want a dynamic experience. As the web browser became a nearly ubiquitous aspect of everyone's operating system, people started developing web-based applications. Considering the rather poor user experience possible within early browsers, this may have seemed like an odd choice, but there are three major benefits to the thin client, at least for the developers.

The barrier of entry for using a web application is very low—a prospective user doesn't need to install any software. If you don't think this matters, ask a decent-sized group how many have used Google Maps (a web-based mapping program); chances are nearly every hand will go up. Ask the same audience how many have used Google Earth (a desktop application that combines satellite imagery with maps that allows people to virtually fly over the earth). Odds are the former will have more takers (especially if you're talking to a group of Mac users—until recently, Google Earth worked only on a PC.) Think of it this way: are you more likely to try an application that involves little more than a click or one that requires you to download and run an executable?

The ubiquity of the browser is closely related to a lower barrier of entry. Unlike most installed applications, a web application will work on any operating system that has a browser written for it. Although this issue is largely mitigated in the Java space, just ask the folks maintaining the Standard Widget Toolkit (SWT) how much fun it is to work with five code bases. A browser-based application allows developers to reach an extremely large audience.

Compared to thick clients, a web app is much easier to maintain. If you've ever worked on a traditional thick app, you know how much fun it is to manage dependencies. Maybe your latest upgrade relies on version 1.2.3 of Spiffy Library but your user only has the 1.2 model. Sure, you can update that as part of your install, but what happens when their other critical application relies on 1.2 and just can't seem to function right with 1.2.3? Of course your users get the rich experience that an installed application provides, but the cost of managing upgrades can be huge. With a browser-based application, we control the servers, so rolling out a fix is (usually) easy. If we need a newer library, we can just add it because we control the deployment environment. This also allows us to make changes more frequently—just push the new code to the servers, and your customers get the latest version.

Despite the advantages, the web has a major downside as an application medium. First, no one would confuse the average web application with Word or Quicken. Sure, if you're moving from the mainframe to the browser, you might not mind, but if your previous app had all the power of the thick client, they might be a tad upset when you take that rich experience away from them. Still, the pros usually outweigh the cons, and a good number of software engineers have spent the last several years building web applications.

Still, users weren't entirely satisfied with the thin client. We may have convinced them to just accept it, but truth be told, the differences between the average web app and a dumb terminal connected to the mainframe are mostly cosmetic.[1] Over the years, we've evolved from using CGI bin scripts to JSPs, and more recently, we have a host of XML-based languages such as XML User Interface Language (XUL) and Extensible Application Markup Language (XAML) that aim to provide a near desktop–like look and feel. Of course Flash has been used to create some fantastic interfaces, but it requires expensive tools and browser plug-ins.

But to reach the widest possible audience, we've been forced to stick pretty close to the browser, meaning that our applications have been tied directly to the synchronous nature of the request/response underpinnings of the Internet. Although request/response makes perfect sense for publishing an article, repainting the entire page even if only a couple of things have changed is not the most usable (or performant) approach. If only we could send a request asynchronously and update just what changed….

1. When a former employer first ventured into the land of web applications in 1998, our CIO was fond of calling our new web apps "lipstick on a pig."

And Then There Was Ajax

Today we have another tool to create truly rich browser-based applications: Ajax. Before you ask, Ajax is more of a technique than a specific technology, though JavaScript is a primary component. We know you're saying, "JavaScript is not worth it," but application and testing frameworks are easing the burden on developers because of the resurgent interest in the language because of Ajax and better tool support. With the introduction of Atlas, Microsoft is throwing its weight firmly behind Ajax, and the infamous Rails web framework comes prebuilt with outstanding Ajax support. In the Java space, Sun has added several Ajax components to its BluePrints Solutions Catalog, and any web framework worth its salt has announced at least minimal support for Ajax.

To be honest though, Ajax isn't anything new. In fact, the "newest" technology related to the term—the `XMLHttpRequest` object (XHR)—has been around since Internet Explorer 5 (released in the spring of 1999) as an ActiveX control. What is new is the level of browser support. Originally, the `XMLHttpRequest` object was supported in only Internet Explorer (thus limiting its use), but starting with Mozilla/Firefox 1.0, Opera 7.6, and Safari 1.2, support is widespread. The little-used object and the basic concepts are even covered in a W3C standard: the DOM Level 3 Load and Save Specification. At this point, especially as applications such as Google Maps, Google Suggest, Gmail, Flickr, Netflix, and A9 proliferate, XHR is becoming a de facto standard.

Unlike many of the approaches used before, Ajax works in most modern browsers and doesn't require any proprietary software or hardware. In fact, one of the real strengths of this approach is that developers don't need to learn some new language or scrap their existing investment in server-side technology. Ajax is a client-side approach and can interact with J2EE, .NET, PHP, Ruby, and CGI scripts—it really is server-agnostic. Short of a few minor security restrictions, you can start using Ajax right now, leveraging what you already know.

Who is using Ajax? As mentioned, Google is clearly one of the leading early adopters with several examples of the technology including Google Maps, Google Suggest, and Gmail, to name just a few applications. Yahoo! is beginning to introduce Ajax controls, and Amazon has been adding a number of interesting features of late. One example involves product categories. Amazon has clearly grown beyond its roots as a purveyor of books, and although their tab metaphor has worked for a while, after Amazon created a certain number of stores, it proved impractical. Enter their new design shown in Figure 1-1. Simply hover over the Product Categories tab to display a list of all the different Amazon stores, allowing you to quickly select the one you wish to explore.

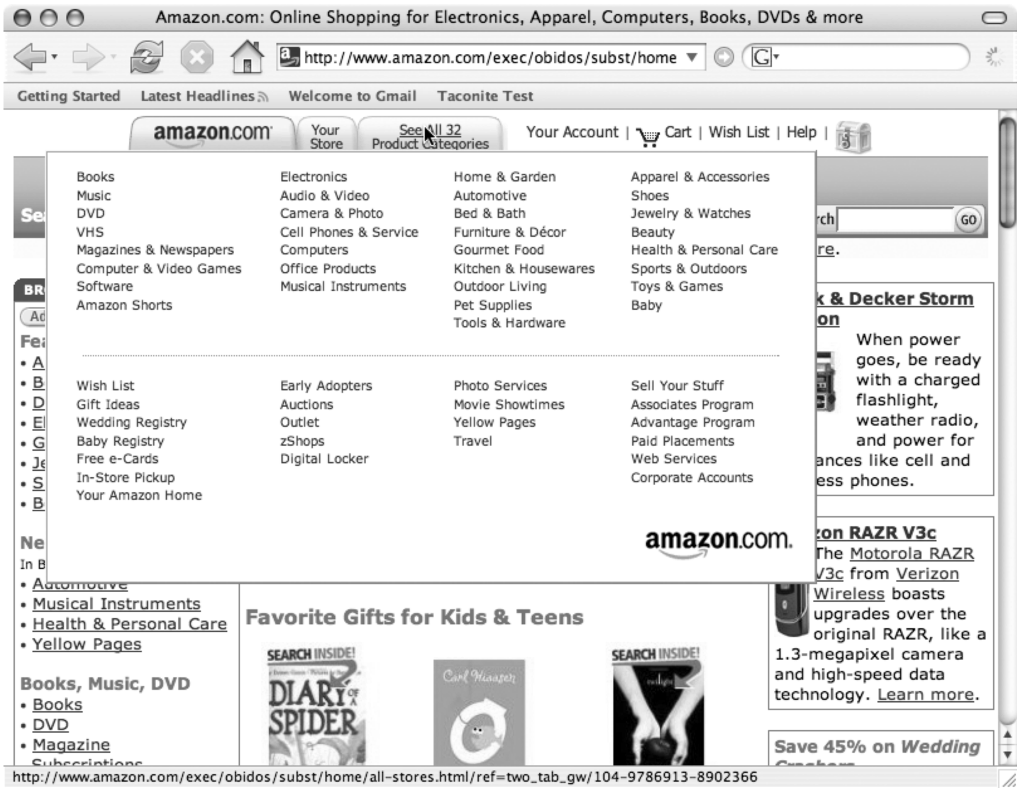
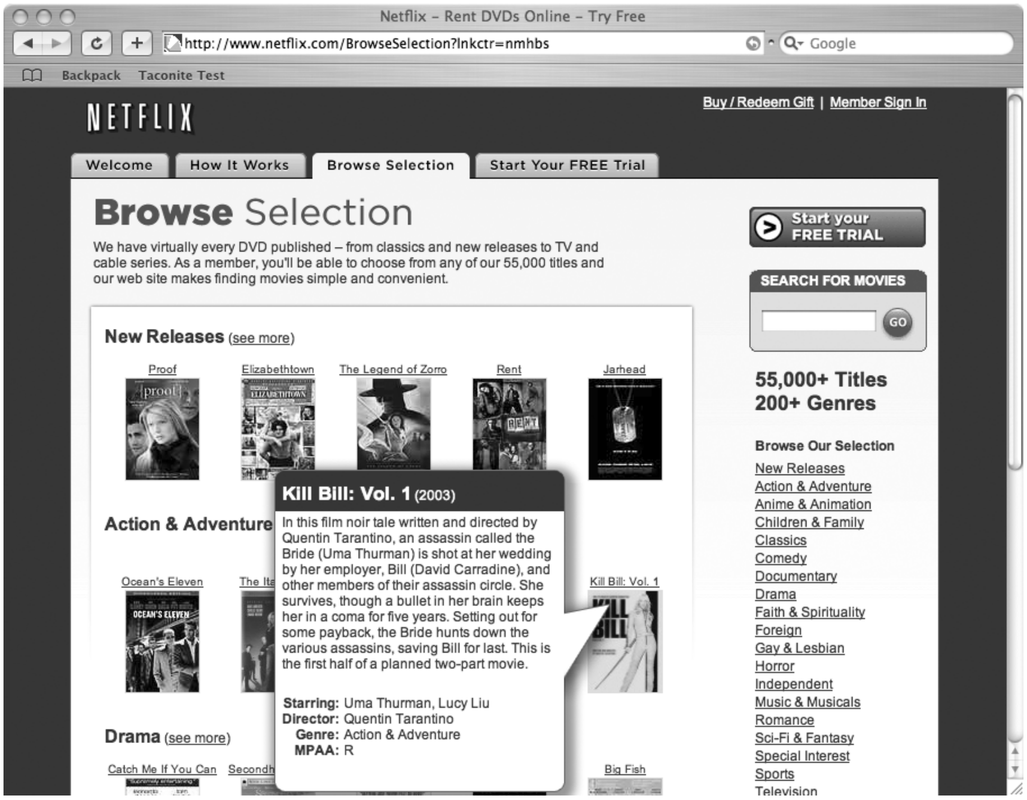


Figure 1-1. *Amazon's product categories*

Another site that takes advantage of a number of Ajax techniques is the DVD rental company Netflix. When a customer hovers over the graphic for a movie, the movie ID is sent back to their central servers, and a bubble appears that provides more details about the movie (see Figure 1-2). Again, the page is not refreshed, and the specifics for each movie aren't found in hidden form fields. This approach allows Netflix to provide more information about its movies without cluttering up its pages. It also makes browsing easier for their customers. They don't have to click the movie and then click back to the list (known as pogo-sticking in the usability community); they simply have to hover over a movie.



Browse Selection

We have virtually every DVD published – from classics and new releases to TV and cable series. As a member, you'll be able to choose from any of our 55,000 titles and our web site makes finding movies simple and convenient.

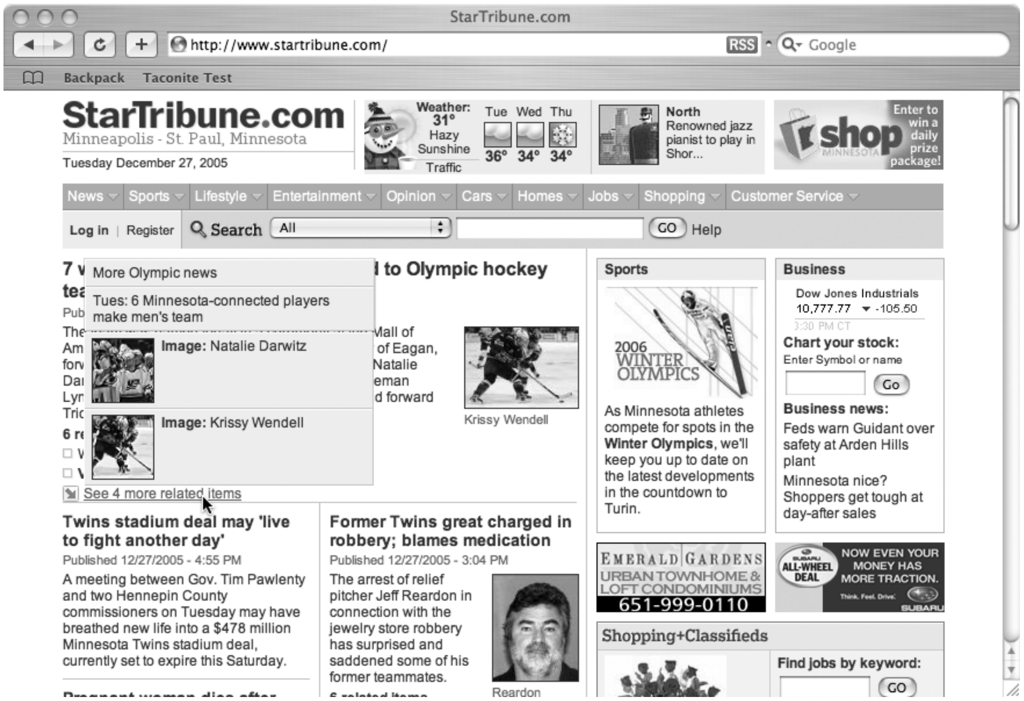
Kill Bill: Vol. 1 (2003)

In this film noir tale written and directed by Quentin Tarantino, an assassin called the Bride (Uma Thurman) is shot at her wedding by her employer, Bill (David Carradine), and other members of their assassin circle. She survives, though a bullet in her brain keeps her in a coma for five years. Setting out for some payback, the Bride hunts down the various assassins, saving Bill for last. This is the first half of a planned two-part movie.

Starring: Uma Thurman, Lucy Liu
Director: Quentin Tarantino
Genre: Action & Adventure
MPAA: R

Figure 1-2. *The Netflix browse feature*

We want to stress that Ajax isn't limited to "dot-com" darlings; corporate developers are starting to scratch the surface as well, with many using Ajax to solve particularly ugly validation situations or to retrieve data on the fly. Heck, one of our local papers, the *Star Tribune* (`www.startribune.com`) recently added a useful Ajax feature. Although most news sites show related articles, there is only so much real estate that can be taken up with these links. Rather than deny their readers these other links, the *Star Tribune* site shows additional related links when a user hovers their mouse over a "See more related items" link (see Figure 1-3).



Tues: 6 Minnesota-connected players make men's team

Image: Natalie Darwitz

Image: Krissy Wendell

7 v... to Olympic hockey

...Mall of ... of Eagan, Natalie ...eman ...d forward

As Minnesota athletes compete for spots in the **Winter Olympics**, we'll keep you up to date on the latest developments in the countdown to Turin.

Dow Jones Industrials 10,777.77 ▼ -105.50

3:30 PM CT

Chart your stock: Enter Symbol or name Go

Business news:

Feds warn Guidant over safety at Arden Hills plant

Minnesota nice? Shoppers get tough at day-after sales

Twins stadium deal may 'live to fight another day'

Published 12/27/2005 - 4:55 PM

A meeting between Gov. Tim Pawlenty and two Hennepin County commissioners on Tuesday may have breathed new life into a $478 million Minnesota Twins stadium deal, currently set to expire this Saturday.

Former Twins great charged in robbery; blames medication

Published 12/27/2005 - 3:04 PM

The arrest of relief pitcher Jeff Reardon in connection with the jewelry store robbery has surprised and saddened some of his former teammates.

Find jobs by keyword: GO

Figure 1-3. StarTribune*'s related items*

Although it isn't exactly new, the *approach* that is the meat of Ajax is an important shift in the Internet's default request/response paradigm. Web application developers are now free to interact with the server asynchronously, meaning they can perform many tasks that before were limited to thick clients. For example, when a user enters a ZIP code, you can validate it and populate other parts of the form with the city and state; or, when they select "United States", you can populate a state drop-down list. We've been able to mimic these approaches before, but it's much simpler to do with Ajax.

So who invented Ajax? The exact origin involved is a subject of debate; however, Jesse James Garrett of Adaptive Path first coined the term in February 2005. In his essay "Ajax: A New Approach to Web Applications" (`www.adaptivepath.com/publications/essays/archives/000385.php`), Garrett discusses how the gap is closing between thick client, or desktop, applications and thin client, or web, applications. Of course, Google really gave the techniques a high profile when it released Google Maps and Google Suggest in Google Labs; also, there have been numerous articles on the subject. But Garrett gave us a term that wasn't quite as wordy as Asynchronous, XMLHttpRequest, JavaScript, CSS, the DOM, and so on. Though originally considered an acronym for Asynchronous JavaScript + XML, the term is now used simply to encompass all the technologies that allow a browser to communicate with the server without refreshing the current page.

We can hear you saying, "So what's the big deal?" Well, using XHR and the fact that you can now work asynchronously with the server lets you create web applications that are far more dynamic. For example, say you have a drop-down list that is filled based on the input in some other field or drop-down list. Ordinarily, you would have to send all the data down to the client when the page first loaded and use JavaScript to populate your drop-down list based on the input. It's not hard to do, but it does bloat the size of your page, and depending on just how dynamic that drop-down list is, size could be an issue. With Ajax, when the trigger field changes or the focus is lost, you can make a simple request to the server for only the information you need to update your drop-down list.

Imagine the possibilities for validation alone. How many times have you written some JavaScript validation logic? Although the edit might be simple in Java or C#, the lack of decent debuggers, combined with JavaScript's weak typing, can make writing them in JavaScript a real pain and error prone to boot. How often do these client-side validation rules duplicate edits on the server? Using XHR, you can make a call to the server and fire *one* set of validation rules. These rules can be far richer and more complex than anything you would write in client-side JavaScript, and you have the full power of debuggers and integrated development environments.

We can hear some of you now saying, "I've been doing that for years with IFRAMES or hidden frames. We've even used this particular technique as a way to post or refresh parts of a page instead of the entire browser, and truth be told, it works." A fair point possibly, but many would consider this approach a hack to get around XHR's original lack of cross-browser support. The XHR object that is the heart of Ajax is truly designed to allow asynchronous retrieval of arbitrary data from the server.

As we've discussed, traditional web applications follow a request/response paradigm. Without Ajax, the entire page (or with IFRAMEs, parts of the page) is reloaded with each request. The previously viewed page is reflected in the browser's history stack (though if IFRAMEs are used, clicking the Back button doesn't always result in what the user expects). However, requests made with XHR are *not* recorded in the browser's history. This too can pose an issue if your users are used to using the Back button to navigate within your web application.

The XMLHttpRequest Object

Although Ajax is more of a technique than a technology, without widespread support for `XMLHttpRequest`, Google Suggest and Ta-da List wouldn't exist as we currently know them. And you wouldn't be reading this book! Chances are pretty good that you won't spend much time working directly with XHR (unless you decide to write your own library), but for much the same reason that introductory programming courses typically use text editors and the command line, this section strips away the fluff to show you what's under the hood.

`XMLHttpRequest` was originally implemented in Internet Explorer 5 as an ActiveX component, meaning that most developers shied away from using `XMLHttpRequest` until its recent adoption as a de facto standard in Mozilla 1.0 and Safari 1.2. It's important to note that `XMLHttpRequest` is *not* a W3C standard, though much of the functionality is covered in a new proposal: the *Document Object Model (DOM) Level 3 Load and Save Specification* (`www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407`). Because it is not a standard, its behavior may differ slightly from browser to browser, though most methods and properties are widely supported. Currently, Firefox, Safari, Opera, Konqueror, and Internet Explorer all implement the behavior of the `XMLHttpRequest` object similarly.

That said, if a significant number of your users still access your site or application with older browsers, you will need to consider your options. If you are going to use Ajax techniques, you will need to either develop an alternative site or allow your application to degrade gracefully. With most usage statistics[2] indicating that only a small fraction of browsers in use today lack `XMLHttpRequest` support, the chances of this being a problem are slim. However, you will need to check your web logs and determine what clients your customers are using to access your sites.

You must first create an `XMLHttpRequest` object using JavaScript before you can use it to send requests and process responses. Since `XMLHttpRequest` is not a W3C standard, creating an instance of `XMLHttpRequest` object requires a browser check. Internet Explorer implements `XMLHttpRequest` as an ActiveX object,[3] and other browsers such as Firefox, Safari, and Opera implement it as a native JavaScript object. Because of these differences, the JavaScript code must contain logic to create an instance of `XMLHttpRequest` using the ActiveX technique or using the native JavaScript object technique.

The previous statement might send shivers down the spines of those who remember the days when the implementation of JavaScript and the DOM varied widely among browsers. Fortunately, in this case you don't need elaborate code to identify the browser type to know how to create an instance of the `XMLHttpRequest` object. All you need to do is check the browser's support of ActiveX objects. If the browser supports ActiveX objects, then you create the `XMLHttpRequest` object using ActiveX. Otherwise, you create it using the native JavaScript object technique. Listing 1-1 demonstrates the simplicity of creating cross-browser JavaScript code that creates an instance of the `XMLHttpRequest` object.

2. `www.w3schools.com/browsers/browsers_stats.asp`
3. In IE 7, XHR will be implemented as a native object, which means all of our checks for ActiveX will, after a while, be the 20 percent case instead of the 80 percent.

Listing 1-1. *Creating an Instance of the* `XMLHttpRequest` *Object*

```
var xmlHttp;
function createXMLHttpRequest() {
    if (window.ActiveXObject) {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else if (window.XMLHttpRequest) {
        xmlHttp = new XMLHttpRequest();
    }
}
```

As you can see, creating the `XMLHttpRequest` object is rather trivial. First, you create a globally scoped variable named `xmlHttp` to hold the reference to the object. The `createXMLHttpRequest` method does the work of actually creating an instance of `XMLHttpRequest`. The method contains simple branching logic that determines how to go about creating the object. The call to `window.ActiveXObject` will return an object or null, which is evaluated by the `if` statement as true or false, thus indicating whether the browser supports ActiveX controls and thus is Internet Explorer. If so, then the `XMLHttpRequest` object is created by instantiating a new instance of `ActiveXObject`, passing a string indicating the type of ActiveX object you want to create. In this instance, you provide `Microsoft.XMLHTTP` to the constructor, indicating your desire to create an instance of `XMLHttpRequest`.

If the call to `window.ActiveXObject` fails, then the JavaScript branches to the `else` statement, which determines whether the browser implements `XMLHttpRequest` as a native JavaScript object. If `window.XMLHttpRequest` exists, then an instance of `XMLHttpRequest` is created, and on the off chance that your user isn't using a modern browser, well, the variable will be undefined.

Thanks to JavaScript's dynamically typed nature and to the fact that `XMLHttpRequest` implementations are compatible across various browsers, you can access the properties and methods of an instance of `XMLHttpRequest` identically, regardless of the method used to create the instance. This greatly simplifies the development process and keeps the JavaScript free of browser-specific logic.

Methods and Properties

Table 1-1 shows some typical methods on the `XMLHttpRequest` object. Don't worry; we'll talk about these methods in greater detail shortly.

Table 1-1. *Standard* `XMLHttpRequest` *Operations*

Method	Description
`abort()`	Stops the current request.
`getAllResponseHeaders()`	Returns all the response headers for the HTTP request as key/value pairs.
`getResponseHeader("header")`	Returns the string value of the specified header.
`open("method", "url")`	Sets the stage for a call to the server. The method argument can be GET, POST, or PUT. The `url` argument can be relative or absolute. This method includes three optional arguments.
`send(content)`	Sends the request to the server.
`setRequestHeader("header", "value")`	Sets the specified header to the supplied value. `open` must be called before attempting to set any headers.

Let's take a closer look at these methods.

`void open(string method, string url, boolean asynch, string username, string password)`: This method sets up your call to the server. This method is meant to be the script-only method of initializing a request. It has two required arguments and three optional arguments. You are required to supply the specific method you are invoking (GET, POST, or PUT) and the URL of the resource you are calling. You may optionally pass a Boolean indicating whether this call is meant to be asynchronous. The default is `true`, which means the request is asynchronous in nature. If you pass a `false`, processing waits until the response returns from the server. Since making calls asynchronously is one of the main benefits of using Ajax, setting this parameter to `false` somewhat defeats the purpose of using the `XMLHttpRequest` object. That said, you may find it useful in certain circumstances such as validating user input before allowing the page to be persisted. The last two parameters are self-explanatory, allowing you to include a specific username and password.

`void send(content)`: This method actually makes the request to the server. If the request was declared as asynchronous, this method returns immediately, otherwise it waits until the response is received. The optional argument can be an instance of a DOM object, an input stream, or a string. The content passed to this method is sent as part of the request body.

`void setRequestHeader(string header, string value)`: This method sets a value for a given header value in the HTTP request. It takes a string representing the header to set and a string representing the value to place in the header. Note that it must be called after a call to `open(…)`.

`void abort()`: This method is really quite self-explanatory; it stops the request.

`string getAllResponseHeaders()`: The core functionality of this method should be familiar to web application developers. It returns a string containing response headers from the HTTP request. Headers include `Content-Length`, `Date`, and `URI`.

`string getResponseHeader(string header)`: This method is a companion to `getAllResponseHeaders()` except it takes an argument representing the specific header value you want, returning this value as a string.

Of all these methods, the two you will use the most are `open(…)` and `send(…)`. The `XMLHttpRequest` object has a number of properties that prove themselves quite useful while designing Ajax interactions.

In addition to these standard methods, the `XMLHttpRequest` object exposes the properties listed in Table 1-2. You'll use these properties extensively when working with `XMLHttpRequest`.

Table 1-2. *Standard `XMLHttpRequest` Properties*

Property	Description
`onreadystatechange`	The event handler that fires at every state change (every time the `readyState` attribute changes); typically a call to a JavaScript function
`readyState`	The state of the request. The five possible values are 0 = uninitialized, 1 = loading, 2 = loaded, 3 = interactive, and 4 = complete
`responseText`	The response from the server as a string
`responseXML`	The response from the server as XML; can be parsed and examined as a DOM object
`status`	The HTTP status code from the server (200 for OK, 404 for Not Found, etc.)
`statusText`	The text version of the HTTP status code (OK or Not Found, etc.)

An Example Interaction

At this point, you might be wondering what a typical Ajax interaction looks like. Figure 1-4 shows the standard interaction paradigm in an Ajax application.

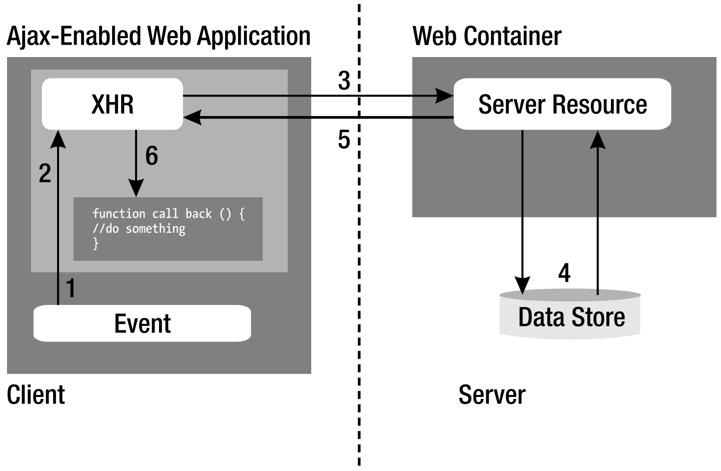


Figure 1-4. *Standard Ajax interaction*

Unlike the standard request/response approach found in a standard web client, an Ajax application does things a little bit differently.

1. A client-side event triggers an Ajax event. Any number of things can trigger this, from a simple onchange event to some specific user action. You might have code like this:

```
<input type="text" d="email" name="email" onblur="validateEmail()";>
```

2. An instance of the `XMLHttpRequest` object is created. Using the `open` method, the call is set up. The URL is set along with the desired HTTP method, typically GET or POST. The request is actually triggered via a call to the `send` method. This code might look something like this:

```
var xmlHttp;
function validateEmail() {
  var email = document.getElementById("email");
  var url = "validate?email=" + escape(email.value);
  if (window.ActiveXObject) {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else if (window.XMLHttpRequest) {
    xmlHttp = new XMLHttpRequest();
  }
xmlHttp.open("GET", url);
xmlHttp.onreadystatechange = callback;
xmlHttp.send(null);
}
```

3. A request is made to the server. This might be a call to a servlet, a CGI script, or any server-side script (written in, say, PHP or ASP.NET).

4. The server can do anything you can think of, including access a data store or even another system (say, the billing system or the HR system).

5. The request is returned to the browser. The `Content-Type` is set to "text/xml"—the `XMLHttpRequest` object can process results only of the "text/html" type. In more complex instances, the response might be quite involved and include JavaScript, DOM manipulation, or other related technologies. Note that you also need to set the headers so that the browser will not cache the results locally. You do this with the following code:

   ```
   response.setHeader("Cache-Control", "no-cache");
   response.setHeader("Pragma", "no-cache");[4]
   ```

6. In this example, you configure the `XMLHttpRequest` object to call the function `callback()` when the processing returns. This function checks the `readyState` property on the `XMLHttpRequest` object and then looks at the status code returned from the server. Provided everything is as expected, the `callback()` function does something interesting on the client. A typical callback method looks something like this:

   ```
   function callback() {
     if (xmlHttp.readyState == 4) {
       if (xmlHttp.status == 200) {
           //do something interesting here
       }
     }
   }
   ```

As you can see, this is different from the normal request/response paradigm but not in a way that is foreign to web developers. Obviously, you have a bit more going on when you create and set up an `XMLHttpRequest` object and when the "callback" has some checks for states and statuses. Typically, you will wrap these standard calls into a library that you will use throughout your application, or you will use one that is available on the web. This arena is new, but a considerable amount of activity is happening in the open source community.

In general, the various frameworks and toolkits available on the web take care of the basic wiring and the browser abstractions, but others add user interface components. Some are purely client-based; others require work on the server. Many of these frameworks

4. Don't `Pragma` and `Cache-Control` do the same thing? Yes, they do, but `Pragma` is defined for backward compatibility.

have just begun development or are in the early phases of release; the landscape is constantly changing, with new libraries and versions coming out regularly. As the field matures, the best ones will become apparent. Some of the more mature libraries include Prototype, script.aculo.us, Dojo Toolkit, Direct Web Remoting (DWR), Taconite, and Rico. This is a dynamic space, so keep your RSS aggregator tuned to those sites dedicated to posting about all things Ajax!

WILL AJAX MAKE MY APPLICATION WEB 2.0?

Since Tim O'Reilly[5] first coined the term, some people have been trying to rebrand their web applications as Web 2.0. What it means to be Web 2.0 is somewhat fluid though. In general, it signals applications that encourage a different style of user participation than those of the past. Web 2.0 is characterized by applications such as Wikipedia and Flickr and activities such as blogging and tagging.

Not everyone is convinced that Web 2.0 is a valuable concept, and some pundits have gone so far as to promise to never use the word again.[6] Although some of the criticism is valid (and some of the hype reminiscent of the late 1990s), many current web applications have characteristics that distinguish them from their older brethren. Many modern applications are more open and participative—note the many mashups (combining web services to create something new) that have been created off of Google Maps such as HousingMaps.[7] In fact, the Google Maps Mania[8] blog has sprung up to track the various applications built on top of Google Maps.

Many of these new-breed applications do indeed use Ajax to some extent, and it is an important component of Web 2.0; however, simply adding some fancy UI widgets doesn't necessarily equate to a Web 2.0 app. Although the boundaries are fuzzy, Web 2.0 is distinguished by using the intelligence in crowds (like Amazon's suggest functionality), software as service (such as Google and Salesforce.com), lightweight development (think deploying daily … or more frequently), along with similar characteristics. Although Ajax is certainly a key component of delivering a richer user experience, it alone is not Web 2.0.

Avoiding Common Gotchas

Ajax really does have the ability to drastically improve the user experience. However, there are a few gotchas that you need to look out for. You may not run into more than a couple of these issues, but before you start using Ajax everywhere, you should keep the following in mind:

5. www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
6. www.joelonsoftware.com/items/2005/10/21.html
7. www.housingmaps.com/
8. http://googlemapsmania.blogspot.com/

Unlinkable pages: You may have noticed that in most of the figures we've shown you, the address bar doesn't change even when the page does. When you use the `XMLHttpRequest` object to communicate with the server, you never need to modify the URL displayed in the address bar. Although this may actually be a plus in some web applications, it also means your users cannot bookmark your page or send the URL to their friends (think about maps or driving directions). This isn't insurmountable; in fact, Google Maps now includes a "Link to this page" link (see Figure 1-5). If links are key to your application or site, be aware that Ajax makes this a bit of a challenge (some frameworks, such as Dojo, provide solutions to this issue).

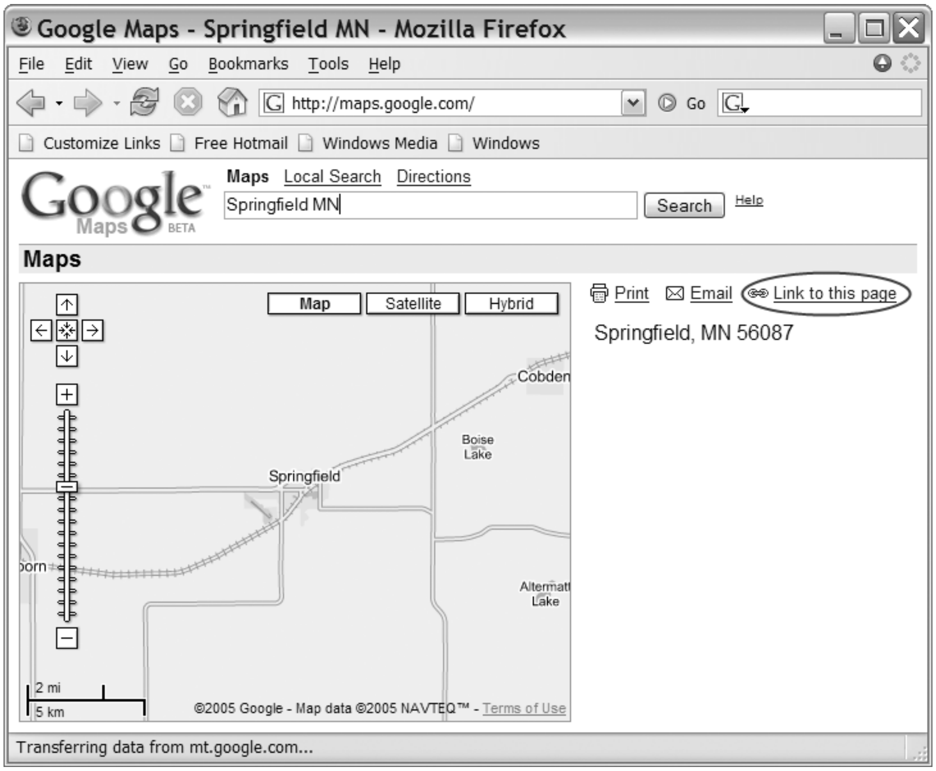


Figure 1-5. *Google Maps "Link to this page" link*

Asynchronous changes: Talking to the server asynchronously is one of the real steps forward with Ajax; however, it isn't without its issues. We've talked about this a few times, but it's worth discussing again: users have been trained to expect the entire page to be repainted anytime things change, so they may not notice when you update just parts of the page. Just because you can reload parts of the page doesn't mean this is the right approach for your entire application—use this approach judiciously. Be careful too that you don't have multiple overlapping asynchronous requests to the server. If you haven't properly coded your client, you may get some pretty odd responses if the server response isn't exactly what you expect (or got during testing).

Lack of visual cues: Since the entire page doesn't repaint, users may not perceive that anything has changed. Ultimately, this is why the Fade Anything Technique (FAT, discussed later in this chapter) was created, but you do have other options. For instance, Gmail uses a "Loading" icon to indicate that it is doing some work (see Figure 1-6). Depending on your application, you may have to add some sort of indication so your users know what is happening.

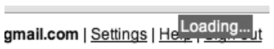
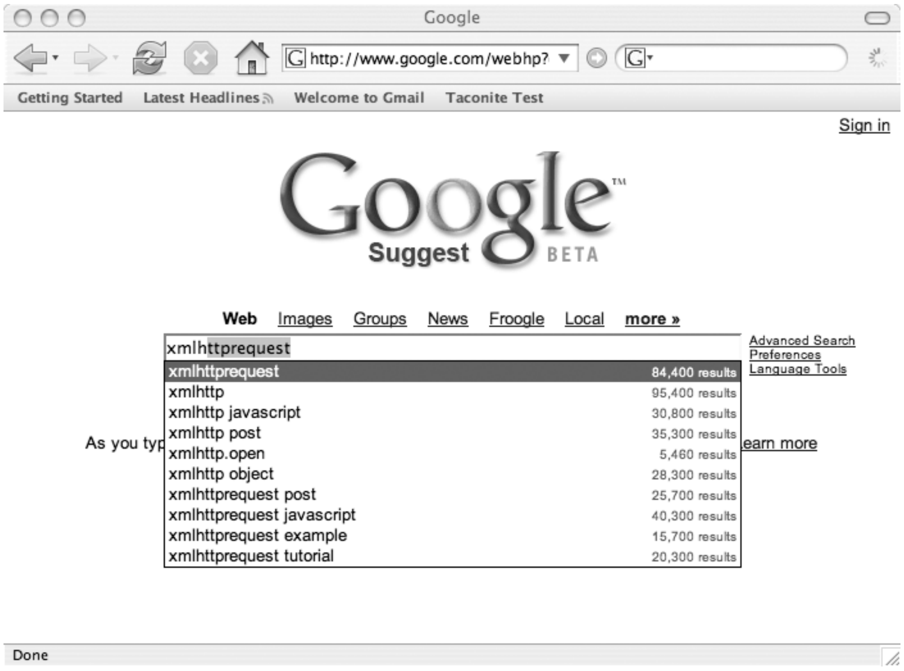


Figure 1-6. *Gmail's "Loading" icon*

The broken Back button: Some web applications deliberately disable the browser's Back button, but few websites do. Of course, with Ajax, clicking the Back button isn't going to do much of anything. If your users are expecting the Back button to work and you're using Ajax to manipulate parts of the page, you may have some problems to solve. (Once again, frameworks such as Dojo provide relief for this issue.)

Code bloat: Never forget that the JavaScript that powers Ajax applications runs locally on your client. Although many developers have powerful machines with reams of random access memory (RAM), some users still have older machines that just don't offer this horsepower. If you put too much JavaScript into your application, you may find sluggish response times on the client side. Even if the JavaScript runs fine, more JavaScript means larger and larger pages, which means longer download times. Until we all have broadband and dual-processor computers, keep JavaScript to a minimum.

Death by a thousand cuts: With the ease of making asynchronous calls, Ajax applications can get a bit chatty. (Remember the early days of entity beans?) You shouldn't add Ajax simply for the sake of adding Ajax. You need to think about each call you make to the server. Making a large number of fine-grain calls can have rather interesting impacts on your server architecture, so you'll want to spend some quality time with your favorite load-testing tool. Autocomplete (see Figure 1-7), although one of the most compelling Ajax widgets, has the potential to be very chatty. Use with caution.



xmlhttprequest

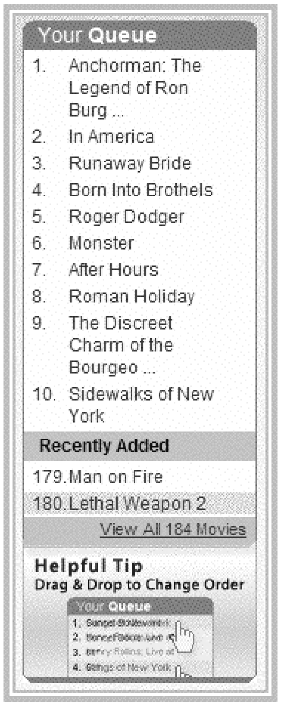
xmlhttprequest	84,400 results
xmlhttp	95,400 results
xmlhttp javascript	30,800 results
xmlhttp post	35,300 results
xmlhttp.open	5,460 results
xmlhttp object	28,300 results
xmlhttprequest post	25,700 results
xmlhttprequest javascript	40,300 results
xmlhttprequest example	15,700 results
xmlhttprequest tutorial	20,300 results

Figure 1-7. *Google Suggest is an example of autocomplete.*

Exposing the business layer: Never forget that your JavaScript is transmitted to the client, and although you can certainly obfuscate the code, if someone really wants to see what you did, they will. With that in mind, be very careful with what you show in your JavaScript. Exposing details about how your server works can open you up to those with ill intent.

Forgetting about security: Despite some arguments to the contrary, Ajax doesn't present any new security vulnerabilities. However, that doesn't mean you can just forget about it. All the standard security issues present in a regular web application still exist in an Ajaxified application.

Breaking established UI conventions: Ajax lets developers create far richer web applications than they've created in the past. However, this doesn't obviate the need to follow normal user interface guidelines. Just because you can do something doesn't mean you should. Also, your snappy new Ajax feature may not be obvious to your users, so don't be afraid to offer a tip like Netflix does with their movie queue (see Figure 1-8). The user can simply drag and drop movies to change their order, but after years of conditioning on web applications, many users might never have realized they could.


Your Queue

1. Anchorman: The Legend of Ron Burg ...
2. In America
3. Runaway Bride
4. Born Into Brothels
5. Roger Dodger
6. Monster
7. After Hours
8. Roman Holiday
9. The Discreet Charm of the Bourgeo ...
10. Sidewalks of New York

Recently Added

179. Man on Fire
180. Lethal Weapon 2

View All 184 Movies

Helpful Tip
Drag & Drop to Change Order

Figure 1-8. *The Netflix queue*[9]

How will you know if you've run afoul of any of these gotchas? We can't stress this enough: test your design with representative users. Before you roll out some snappy new Ajax feature, do some paper mock-ups, and run them by a few users before you spend the time and effort developing it. An hour or two of testing can save you from dealing with larger issues later.

Ajax Patterns

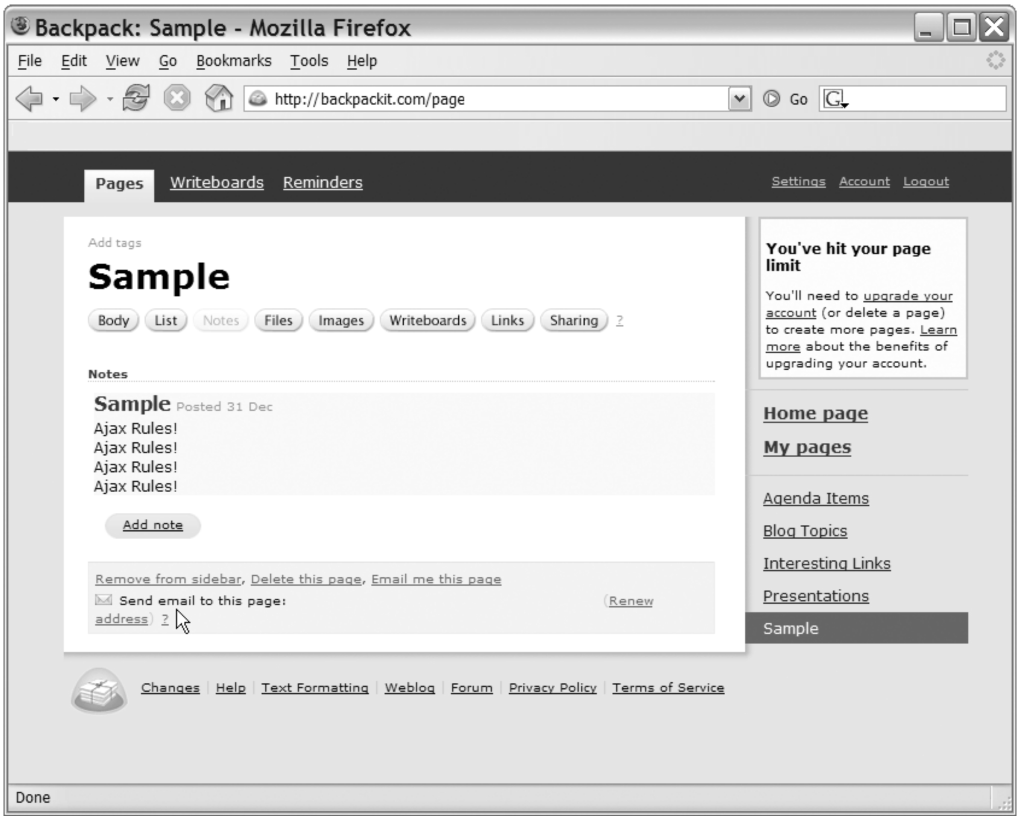
Like any good technology, Ajax already has a slew of patterns. For a detailed look at Ajax patterns, please take a look at *Ajax Patterns and Best Practices* by Christian Gross (Apress, 2006) and the website Ajax Patterns (`http://ajaxpatterns.org/wiki/index.php?title=Main_Page`).

9. `http://ajaxian.com/archives/drag-and-drop-ajaxian-indicators`

The Fade Anything Technique (FAT)

One of the really slick things about Ajax is that you can modify just part of a web page. Rather than repaint the entire view, you can update just the part that changes. Although this is a handy technique, it may confuse users who are expecting a full-page refresh. With this in mind, 37signals uses the Yellow Fade Technique (YFT) in its products like Basecamp (`www.basecamphq.com`) and Backpack (`www.backpackit.com`) as a way of subtly indicating to the user what has changed. YFT does just what it suggests: the part of the page that changed is repainted in yellow, and it slowly fades back to the original background color (see Figure 1-9).

The Fade Anything Technique (FAT) pattern is similar in nature. In essence, the only real change is the color you use to fade; after all, yellow might not be the best option for you. Implementing this technique is not terribly difficult; you can find sample code using your favorite search engine or, better yet, just use one of the libraries that makes this simple, such as script.aculo.us.[10]


Backpack: Sample - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://backpackit.com/page
Go
Pages Writeboards Reminders
Settings Account Logout
Add tags
Sample
Body List Notes Files Images Writeboards Links Sharing ?
Notes
Add note
Home page
My pages
Agenda Items
Blog Topics
Interesting Links
Presentations
Sample
Changes | Help | Text Formatting | Weblog | Forum | Privacy Policy | Terms of Service
Done

Sample Posted 31 Dec
Ajax Rules!
Ajax Rules!
Ajax Rules!
Ajax Rules!

Remove from sidebar, Delete this page, Email me this page
Send email to this page: (Renew address) ?

You've hit your page limit
You'll need to upgrade your account (or delete a page) to create more pages. Learn more about the benefits of upgrading your account.

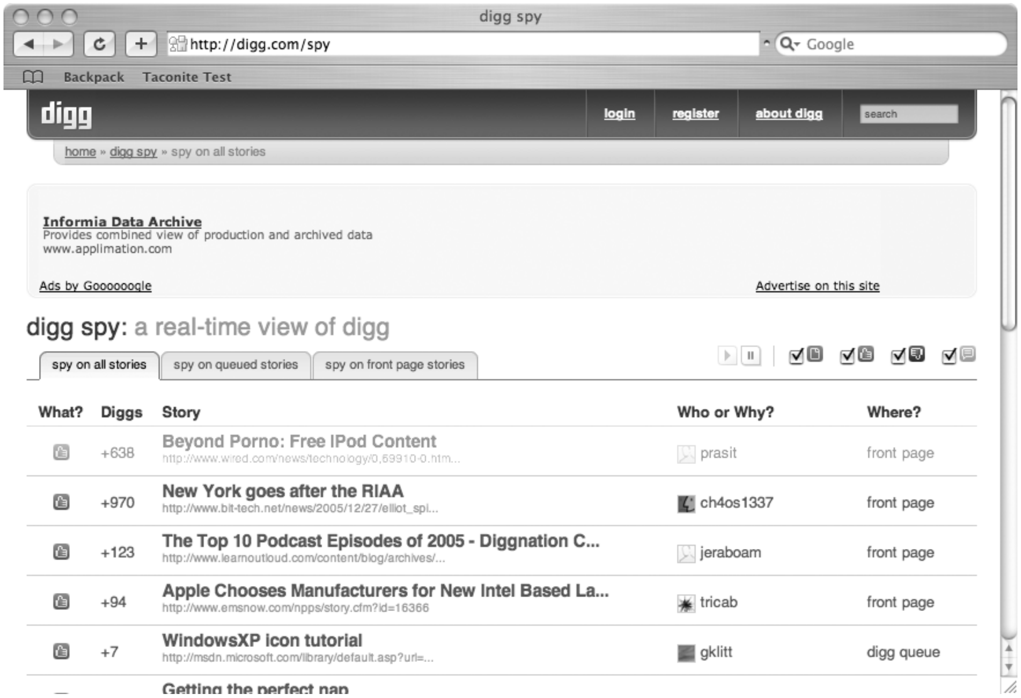
Figure 1-9. *Yellow Fade Technique as seen in 37signals' Backpack*

10. `http://script.aculo.us/`

Auto Refresh

Being able to refresh parts of the page automatically is useful. With weather, news, or other information streams that change over time, repainting just the parts that change makes a lot more sense than refreshing the entire page for a few minor changes. Of course, this might not be terribly obvious to users who are trained to click the Refresh button, which is why the Auto Refresh pattern is so often paired with FAT.

Auto Refresh offers a significant benefit beyond less work for your user: it also reduces the load on your server. Rather than having thousands of users constantly clicking the Refresh button, you can set a specific polling period that should spread out the requests more evenly. Digg (`www.digg.com`) is a news site catering to the technology community that uses social bookmarking and community editorial control to publicize interesting stories. The content is user-driven and changes frequently, making it a perfect place to use Auto Refresh (see Figure 1-10).



Informia Data Archive
Provides combined view of production and archived data
www.applimation.com

digg spy: a real-time view of digg

What?	Diggs	Story	Who or Why?	Where?
	+638	Beyond Porno: Free IPod Content http://www.wired.com/news/technology/0,69910-0.htm...	prasit	front page
	+970	New York goes after the RIAA http://www.bit-tech.net/news/2005/12/27/elliot_spi...	ch4os1337	front page
	+123	The Top 10 Podcast Episodes of 2005 - Diggnation C... http://www.learnoutloud.com/content/blog/archives/...	jeraboam	front page
	+94	Apple Chooses Manufacturers for New Intel Based La... http://www.emsnow.com/npps/story.cfm?id=16366	tricab	front page
	+7	WindowsXP icon tutorial http://msdn.microsoft.com/library/default.asp?url=...	gklitt	digg queue
		Getting the perfect nap		

Figure 1-10. *Auto Refresh at work—new stories appear on the site while you watch!*

Partial Page Paint

We've talked about this point quite a bit, but one of the real strengths of Ajax is that you no longer need to repaint the entire page; instead, you can just modify what has changed.

Clearly, you can use this in conjunction with FAT or Auto Refresh. In fact, this can be helpful for web applications.

Many of the existing frameworks will help you modify part of the page, and thanks to solid DOM support in modern browsers, this approach is much easier than you think. Figure 1-11 shows A9's BlockView (`http://maps.a9.com`) feature, an example of the Partial Page Paint pattern. When you select a different part of the map on the left, the corresponding pictures of the street automatically change to reflect where your map is pointing (assuming pictures exist).

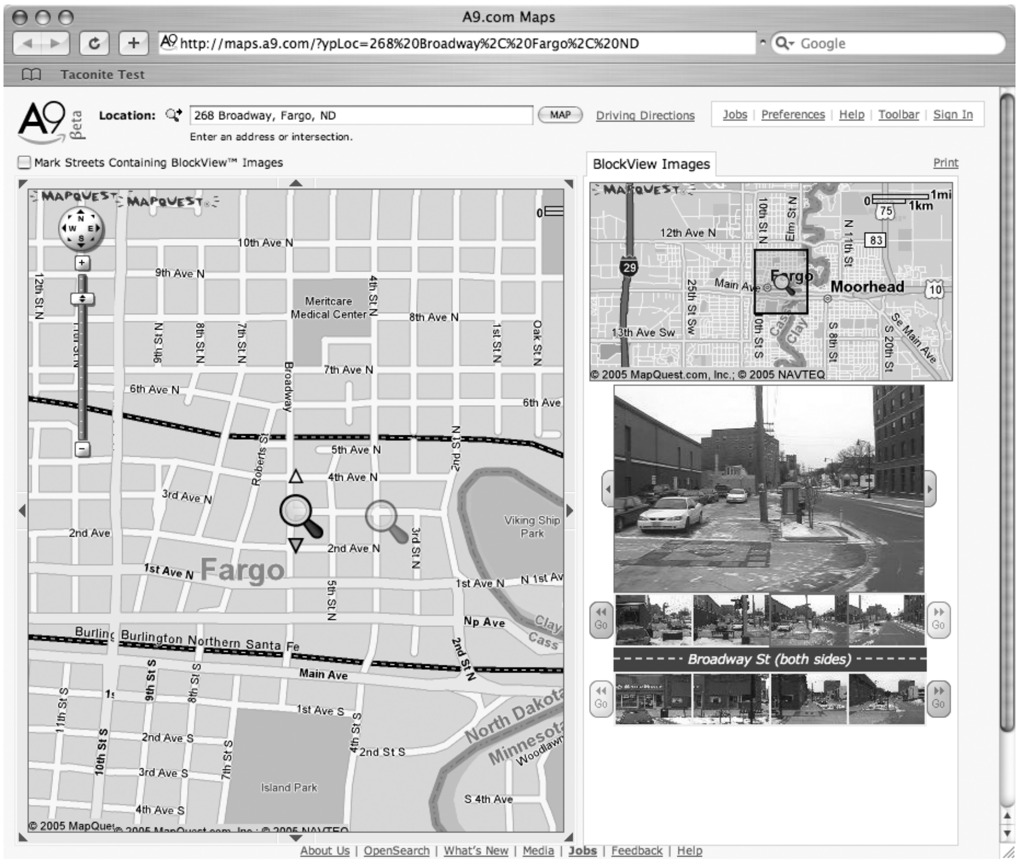
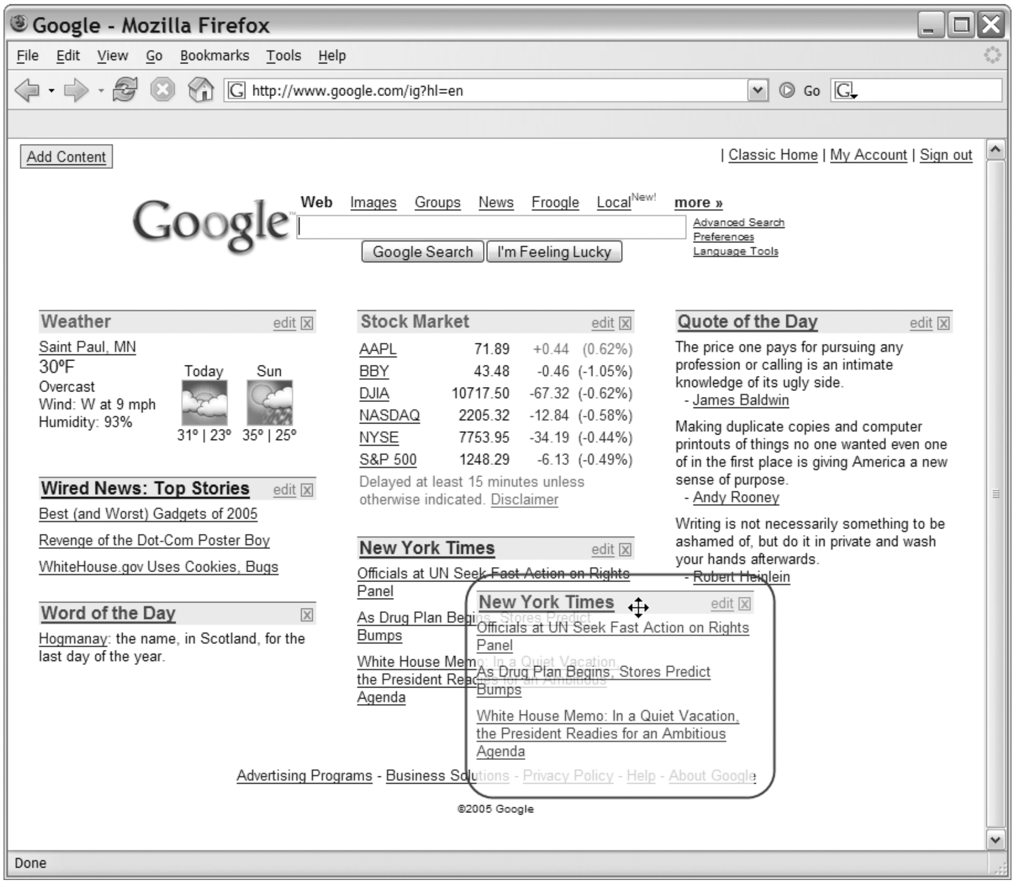

A9.com Maps
http://maps.a9.com/?ypLoc=268%20Broadway%2C%20Fargo%2C%20ND
Google
Taconite Test
Location: 268 Broadway, Fargo, ND
MAP
Driving Directions
Jobs | Preferences | Help | Toolbar | Sign In
Enter an address or intersection.
Mark Streets Containing BlockView™ Images
BlockView Images
Print
Fargo
Moorhead
Broadway St (both sides)
© 2005 MapQuest.com, Inc.; © 2005 NAVTEQ
About Us | OpenSearch | What's New | Media | Jobs | Feedback | Help

Figure 1-11. *A9's BlockView feature*

Draggable DOM

Portals were supposed to be the solution to all our problems. Corporate intranet sites were designed to be one-stop shopping for employees to have all the information they needed at their fingertips. Your most frequently used applications, links to your key reports, industry news—a customized portal was supposed to be the answer. Unfortunately, corporate intranets never really took off, but at least part of the reason had to do with clumsy interfaces for adding new sections and moving existing ones. Typically, you had to go to a separate administration page to make your changes (a full-page refresh), save your changes, and return to your home page (another page refresh). Although this approach worked, it certainly wasn't ideal.

Portals are given a new life with Ajax, especially using the Draggable DOM pattern. With this approach, the individual sections are editable right on the main page, and to customize the page, you simply grab them with your mouse and drag them to their new location. Several sites, including the personalized Google (`www.google.com/ig`), have used this pattern, as shown in Figure 1-12.


Google - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.google.com/ig?hl=en
Go
Add Content
| Classic Home | My Account | Sign out
Web Images Groups News Froogle Local New! more »
Google Search I'm Feeling Lucky
Advanced Search
Preferences
Language Tools
Done

Weather edit
Saint Paul, MN
30ºF
Overcast
Wind: W at 9 mph
Humidity: 93%
Today Sun
31º | 23º 35º | 25º

Stock Market edit

AAPL	71.89	+0.44	(0.62%)
BBY	43.48	-0.46	(-1.05%)
DJIA	10717.50	-67.32	(-0.62%)
NASDAQ	2205.32	-12.84	(-0.58%)
NYSE	7753.95	-34.19	(-0.44%)
S&P 500	1248.29	-6.13	(-0.49%)

Delayed at least 15 minutes unless otherwise indicated. Disclaimer

Quote of the Day edit
The price one pays for pursuing any profession or calling is an intimate knowledge of its ugly side.
- James Baldwin

Making duplicate copies and computer printouts of things no one wanted even one of in the first place is giving America a new sense of purpose.
- Andy Rooney

Writing is not necessarily something to be ashamed of, but do it in private and wash your hands afterwards.
- Robert Heinlein

Wired News: Top Stories edit
Best (and Worst) Gadgets of 2005
Revenge of the Dot-Com Poster Boy
WhiteHouse.gov Uses Cookies, Bugs

Word of the Day
Hogmanay: the name, in Scotland, for the last day of the year.

New York Times edit
Officials at UN Seek Fast Action on Rights Panel
As Drug Plan Begins, Stores Predict Bumps
White House Memo: In a Quiet Vacation, the President Readies for an Ambitious Agenda

Advertising Programs - Business Solutions - Privacy Policy - Help - About Google
©2005 Google

Figure 1-12. *Personalized Google uses Draggable DOM.*

Summary

The Internet has certainly evolved from its early origins as a way for researchers to connect and share information. Though the Internet began with simple textual browsers and static pages, it is now hard to find a company that doesn't have a polished website. In its early days, who could have possibly imagined that people would one day flock to the Internet to research that new car or buy the latest Stephen King novel?

Developers fed up with the difficulty of deploying thick client applications to thousands of users looked to the web to ease their burden. Several web application technologies have been developed over the years, some proprietary, others requiring significant programming abilities. Though some provided a richer user experience than others, no one would confuse a thin client application with its desktop-based cousin. Still, the ease of deployment, the ability to reach a wider customer base, and the lower cost of maintenance means that despite the limitations of browsers, they are still the target platform of choice for many applications.

Developers have used hacks to circumvent some of the most troublesome restrictions the Internet places on developers. Various remote scripting options and HTML elements let developers work asynchronously with the server, but it wasn't until the major browsers added support for the `XMLHttpRequest` object that a true cross-browser method was possible. With companies such as Google, Yahoo!, and Amazon leading the way, we are finally seeing browser-based applications that rival thick clients. With Ajax, you get the best of both worlds. Your code sits on a server that you control, and any customer with a browser can access an application that provides a full, rich user experience.