

Ajax and the Enterprise: Code Freeze 2006

Nathaniel T. Schutta: ntschutta.com

Ajax has certainly garnered an incredible amount of buzz these days but before you dismiss it as just this month's over hyped technology, take a moment to see past the rhetoric. Ajax isn't a golden hammer but then what is? While Ajax won't cure hunger anytime soon, applied properly, it has the potential to significantly improve the usability of our web applications. Like it or not, our customers are being exposed the possibilities of Ajax everyday – it may have (largely) started in the Google collective but it's quickly spreading to less technocratic organizations.

What is Ajax?

Ajax the word dates back about a year to an article written by Jesse James Garrett of Adaptive Path¹. Garrett needed a concise way (and one that would be accessible to "C"-level execs) to describe a host of technologies: XHTML, CSS, interacting with the DOM, XML, XSLT, JavaScript, and the XMLHttpRequest object and thus was born Ajax or Asynchronous JavaScript + XML². You should immediately note that Ajax isn't a specific thing but a collection of technologies none of which is, in fact, new. Why then all the excitement?

Give me an 'A'

Though it has lost its status as a true acronym (some still use the all caps AJAX as a way of distinguishing between Ajax the technology thing and Ajax the football club) remember that the A is Asynchronous. The web as we know it didn't have applications and ecommerce as a design goal – it was a way of linking researchers and their papers together (rather than have to look that cite up, just follow the link to the text). As such, the Internet is designed in a request/response fashion; a user asks for a document and the server returns the entire document.

While this might be fine and dandy for sharing the latest draft of your PhD thesis, it presents some interesting issues for those of us designing web applications. Though the web frees us from much of the hassle of maintaining distributed solutions (posting upgrades to central servers that we control is orders of magnitudes easier than dealing with even just a few hundred machines in the wild – just look at a sample test matrix for IE³). While better than a mainframe "green screen", browser based applications still leave a fair amount to be desired in the user experience realm – few will confuse Quicken with their bank's web presence.

Despite any drawbacks, we convinced our users that the web was the answer for their next application. Along the way, we trained them to accept a full-page

repaint – after all, that’s what the web is about. Sure, we might have used a few hacks here and there but, by and large, if we needed something from the server we sent the whole page, the server returned the whole page, and we in turn rendered the whole page.

Ajax finally frees us from the request/response paradigm. We can send fine-grained requests to the server and get back just the information we want. Using the DOM, we can turn around and update just the bits and pieces that have changed. Using Ajax, we can actually make web applications that rival their desktop-based cousins in terms of look and feel.

But I’ve been doing this for years...

By now a few of you might be saying, hey, I’ve used IFRAMEs for years as a way of posting bits. You know what, so have I. However, there are some significant differences between XHR and iframes. First, an iframe is still just a standard request/response – it’s expecting the entire frame to be rendered. Second, iframes don’t provide any mechanism for figuring out when the request is done – again, it’s just expecting the display the contents in the frame. While you certainly can make “Ajax-like” applications using nothing but iframes (in fact, browser compatibilities may make this a requirement) it just isn’t the same.

In the pre-Ajax era, many have used remote scripting⁴ as a technique to make more responsive applications. Microsoft had their own solution⁵ but of course it requires you to use Microsoft products and also requires Java to be installed on the clients. Brent Ashley⁶ created a client side JS library called JavaScript Remote Scripting⁷ that allowed developers to connect asynchronously to a variety of server side technologies.

What’s in a name?

There is a fair amount of anti-Ajax sentiment in the blogosphere these days. Some object to the name or don’t see what’s new (⁸ and ⁹) There’s even a rather clever spoof of a Nielsen Alertbox¹⁰ called Why Ajax Sucks (Most of the Time)¹¹ that has caused quite a stir. Others actually point out that there are some places we just shouldn’t use Ajax¹². Still, names matter and while some believe Ajax is as meaningless¹³ as Tim O’Reilly’s Web 2.0 meme (see ¹⁴ and ¹⁵) capturing the concepts behind Ajax in a single word helps raise the level of dialog that is possible when it comes to talking about web apps.

While the technology behind Ajax isn’t anything new, the way it’s being applied certainly is. Google Maps¹⁶ is a significant step forward (especially all the mashups¹⁷ (see ¹⁸ and ¹⁹) and just look at what the sharp folks at 37signals²⁰ crank out (seemingly daily). If you’ve never tried Ta-da List²¹ or Backpack²², you really should – 37signals knows how and (more importantly) where to apply Ajax to maximum effect.

Why Now?

We've established that Ajax is just a collection of related technologies, none of which is new – which begs the question, why is it a big deal now? The engine that makes Ajax work today is the XMLHttpRequest object, something that Microsoft created back with IE 5.0²³. After years of fighting against defacto standards of IE, Safari and Firefox decided to support XHR thus freeing a huge segment of the web world to embrace Ajax. My introduction to the topic was a post on Joel on Software²⁴ where he dissected Google Suggest²⁵ - one of the first "Ajax apps".

XHR is a pretty simple object – it has a few methods and a few properties (see ²⁶, ²⁷, and ²⁸ for more information). To create an instance, we have to do a quick browser check:

Listing 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();
  }
}
```

Since IE still implements XHR as an ActiveX component, we need slightly different creation approaches (this changes in IE 7²⁹). Fortunately, that's the end of the browser specific code. Once we have an instance, we go ahead and set a callback function. We do this by setting the `onreadystatechange` property to the JavaScript method we want to act as the event handler for this call. This method will look at the `readyState` property against a set of magic numbers (0-4) indicating the state of the request (uninitialized to complete).

Next, we setup the call to the server using the `open` method, which takes the HTTP method to use in the call (GET or POST) and the URL of the resource to call (along with three optional parameters). At this point, we can pass our request off to the server via the `send` method. The `send` method takes one argument, typically a string or a DOM object – this parameter is transmitted to the destination URL as part of the request body. When providing a parameter to the `send` method, be sure the method assigned in the `open` method is POST. Use null when there is no data to be sent as part of the request body.

The function we defined via the callback function will be called on each state change the XHR object undergoes (so if you put an alert in this function, it will be called 5 times). In general, we only look for 4 since that means our request

is complete. At this point we may check the status of the response via the `status` property – this property is just the http status code³⁰ (i.e., 200 which indicates OK or 204 which means No Content). A simplified example is shown in Listing 2.

Listing 2. The `simpleRequest.html` Page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple XMLHttpRequest</title>

<script type="text/javascript">
var xmlhttp;

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

function startRequest() {
    createXMLHttpRequest();
    xmlhttp.onreadystatechange = handleStateChange;
    xmlhttp.open("GET", "simpleResponse.xml");
    xmlhttp.send(null);
}

function handleStateChange() {
    if(xmlhttp.readyState == 4) {
        if(xmlhttp.status == 200) {
            alert("The server replied with: " + xmlhttp.responseText);
        }
    }
}
</script>
</head>

<body>
    <form action="#">
        <input type="button" value="Start Basic Asynchronous Request"
            onclick="startRequest();"/>
    </form>
</body>
</html>
```

Mocking the Server

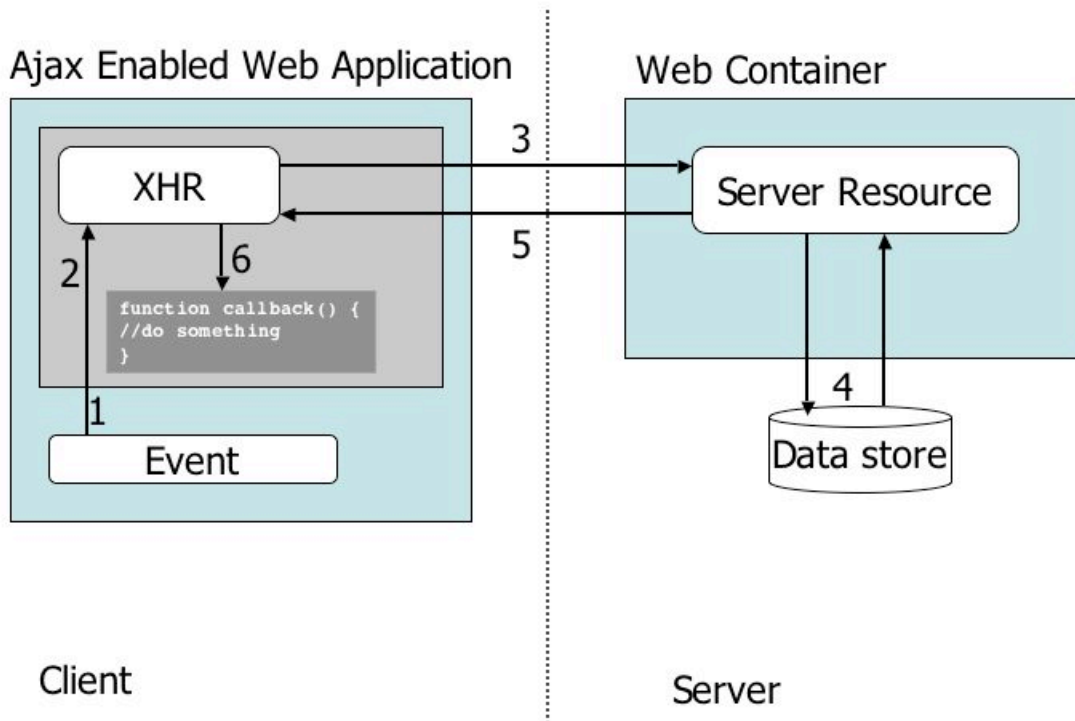
You may have noticed that we weren't actually calling the server here – we just referenced an XML file. While this would be atypical in a production setting, this

can be very useful in a testing situation. Most people are familiar with the ideas behind mock objects (see ³¹, ³² and ³³); with Ajax, we can always mock the server³⁴. This can be very helpful when creating and debugging the JavaScript required for the callback function especially when combined with JsUnit³⁵. This lets you work on your client code independent of the server side, which can increase your productivity (the tasks can be implemented in parallel).

An Ajax Interaction

A typical Ajax interaction looks something like Figure 1.

Figure 1. Standard Ajax interaction



Obviously, this doesn't look a ton different than a standard web application however there are some subtle differences.

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" id="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 technique.
4. The server can do anything you can think of, including accessing a data store or even another 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");

```
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
    }
  }
}

```

Ajax Gotchas

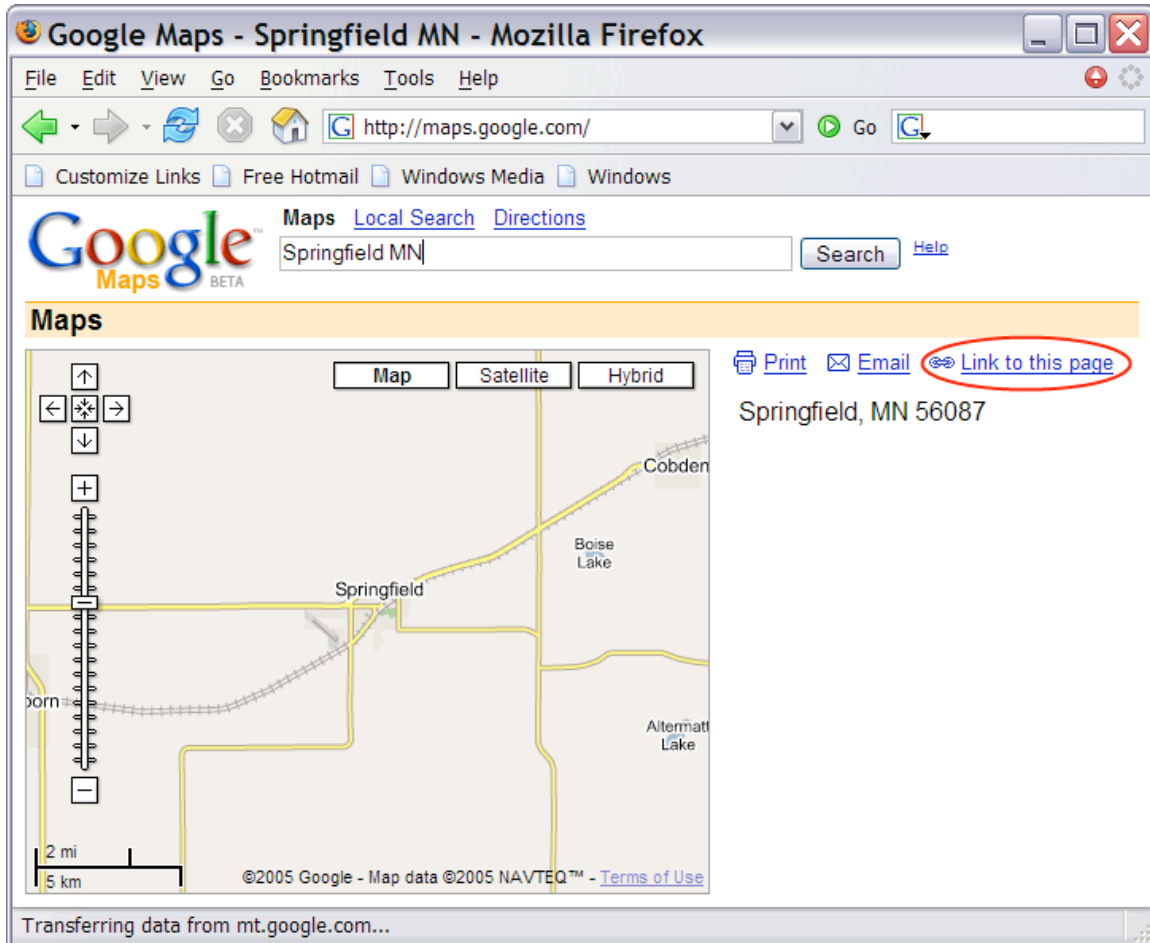
Ajax really does have the ability to drastically improve the user experience however, there are a few gotchas that you need to look out for. You may not run into more than a couple of these issues, but before you start using Ajax everywhere, you should keep the following in mind:

- **Unlinkable Pages**

In most Ajax applications, the address bar doesn't change even when the page does, in fact there has been some discussions³⁶ on single page (like Google Maps) versus multi page web apps (like Gmail). When you use the

XMLHttpRequest object to communicate with the server, you never need to modify the URL displayed in the address bar. While this may actually be a plus in some Web applications, it also means your users cannot bookmark your page or send a URL to their friends (think about maps or driving directions). This isn't insurmountable; in fact, Google Maps includes a Link to This Page link (see Figure 2). If links are key to your application or site, be aware that Ajax makes this a bit a challenge.

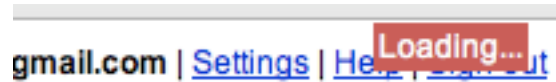
Figure 2. Google Maps Link to This Page link



- **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.
- **Lack of visual cues**
Since the entire page doesn't repaint, users may not perceive that

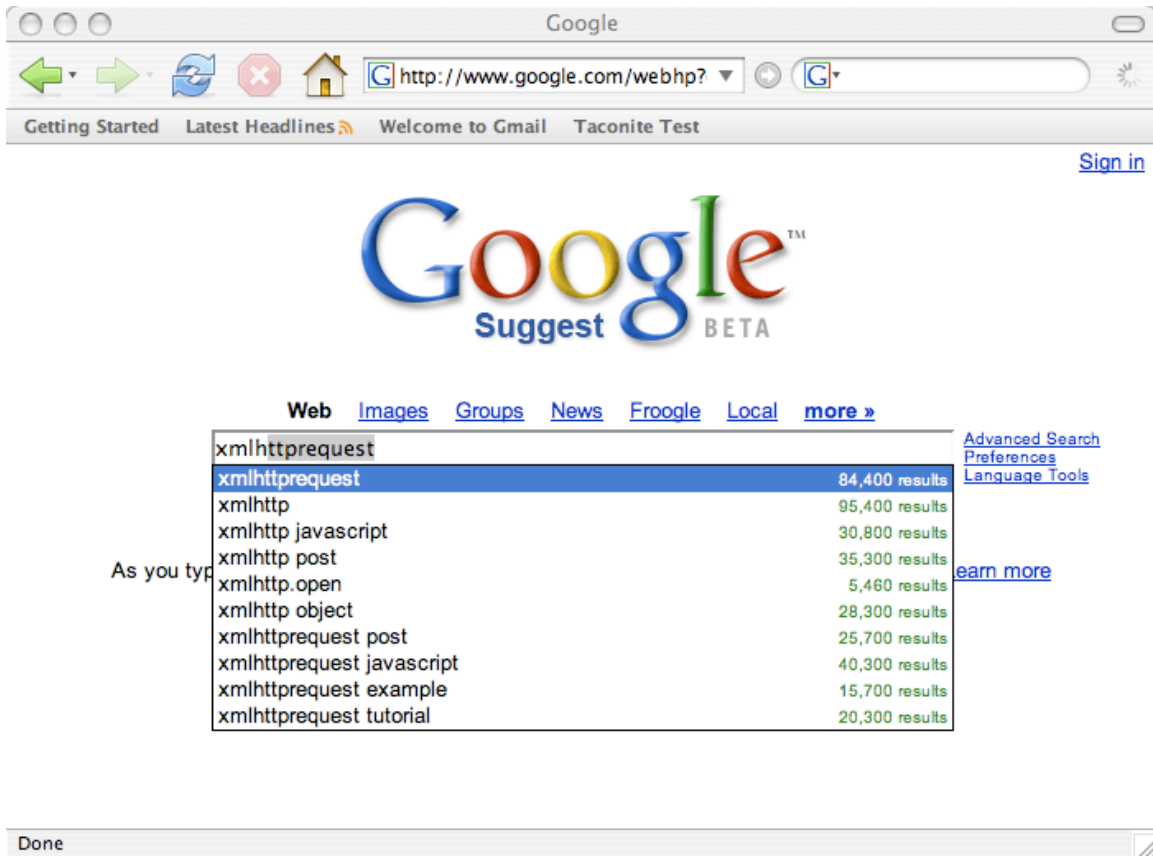
anything has changed. Ultimately, this is why FAT was created, but you do have other options. For instance, Gmail uses a “Loading” icon to indicate that it is doing some work (see Figure 3). Depending on your application, you may have to add some sort of indication so your users know what is happening.

Figure 3. Gmail’s “Loading” icon



- **The broken back button**
Some Web applications deliberately disable the browser’s back button, but few Web sites 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.
- **Code bloat**
Never forget that the JavaScript that powers Ajax applications runs locally on your client. While 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 and 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. Auto complete (see Figure 4), while one of the most compelling Ajax widgets, has the potential to be very chatty – use with caution.

Figure 4. Google Suggest is an example of auto complete.



- **Exposing the business layer:**
Never forget that your JavaScript is transmitted to the client, and while 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, you're 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-x). The user can simply drag and drop movies to change their order but after years of conditioning on web affordances, many users might never have realized they could.

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 larger issues later.

Ajax and you

Is Ajax rocket science? There is a popular myth that Ajax requires PhD candidates from Stanford (see ³⁷, ³⁸, and ³⁹) and while it certainly doesn't hurt to have Ivy League types design your applications, it really isn't needed. The basics of Google Maps can be done in just a few hours – seriously, the guys at Ajaxian⁴⁰ have done it⁴¹!

There is also a sense that Ajax requires a new developer (see ⁴², ⁴³ and ⁴⁴). Honestly, it's quite likely that your current development staff can more than handle Ajax work. You will have to accept JavaScript as a first class citizen though. Many developers have shunned JavaScript – early cross browser support combined with a lack of tools has caused a lot of people to see JS as nothing more than a toy language for graphics designers. However, the rise in popularity of dynamic languages⁴⁵, combined with Java's upcoming support⁴⁶ for Rhino⁴⁷, (and let's not forget about Groovy⁴⁸, Jython⁴⁹, and JRuby⁵⁰) shows that scripting languages are finally being seen as valuable tools.

Tools

Tool support is certainly improving but so far IDEs aren't much help when it comes to JavaScript (or any dynamic language for that matter). This situation will change within the next several months but for now, there are a number of tools that you can use to make developing Ajax easier.

Debugging is often mentioned as the biggest complaint people have when developing JavaScript – alerts just don't cut it. Luckily for us, there are a number of solutions. First, Venkman⁵¹ is an excellent debugger for Firefox. We've also got a host of logging solutions such as Log4Ajax⁵², Lumberjack⁵³, and Log4JS⁵⁴.

Crafting an Ajax application means you'll be modifying the DOM. While it may be pretty obvious where/what all the elements are on your page, sometimes it's very useful to be able to literally inspect the DOM⁵⁵. We've also got a few arrows to help us validate our HTML: HTML Validator⁵⁶ and Checky⁵⁷.

The test drive development⁵⁸ folks will be happy to know that we've got no excuse not to test our JavaScript either. We have a JavaScript port of JUnit called JsUnit⁵⁹ and we can also take advantage of a ThoughtWorks⁶⁰ project called Selenium (⁶¹ and ⁶²).

Libraries and Toolkits

Of course chances are you'll never deal directly with XHR. As you would expect, there are a number of toolkits and frameworks (with more seemingly added daily). The space changes frequently so keep an eye on the Frameworks⁶³ page on Ajax Patterns⁶⁴. Some of the most popular ones include Prototype⁶⁵, Script.aculo.us⁶⁶, Taconite⁶⁷, Dojo⁶⁸ and DWR⁶⁹. As expected, this space changes often so stay tuned.

Getting Started

If you've made it this far, I hope that you're ready to start using Ajax in your applications! By now you should realize that Ajax isn't a golden hammer and it won't cure cancer but it really can help you provide a richer user experience. But don't use Ajax just to say your application has Ajax – if it doesn't make sense then don't do it! Start small; validation is a great first step. Auto complete (aka Google Suggest) is a very powerful feature but be careful, it can lead to a chatty client. Using Ajax, we can essentially lazily instantiate⁷⁰ parts of our page. Rather than download every possible foo, we can just ask for the ones that we need as users make selections.

We can also provide some very rich tool tips as seen with Netflix⁷¹ in their browse section⁷² – simply hover over a cover to see expanded information about a given movie. Oh, and if your site has a shopping cart, there's not reason to force your users to hit recalculate anymore⁷³!

Getting More

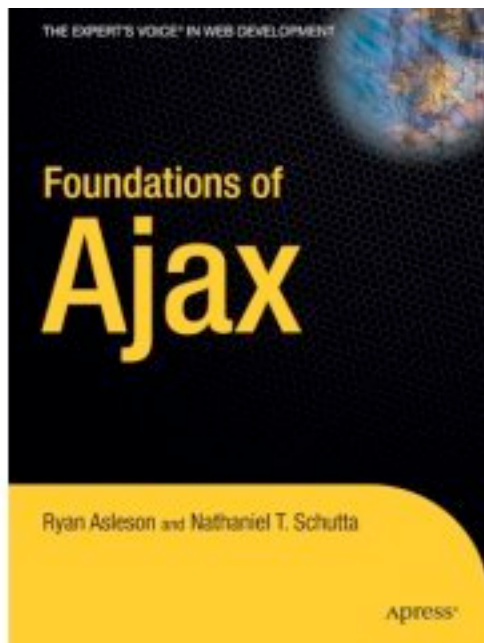
I sure hope this piece has whetted your appetite for all things Ajax! Of course there's only so much I can cover in a few thousand words and I have to recommend my book, Foundations of Ajax⁷⁴, which I co-wrote with Ryan Asleson. As long as I'm in shameless plug mode, I'll recommend my blog: Just a Thought⁷⁵. While I'm not nearly as focused as Ajaxian⁷⁶ (which I check daily), I've written more than a few articles on the topic. I can also recommend Ajax Patterns⁷⁷, Ajax Matters⁷⁸ and Ajax blog⁷⁹. You should keep a close eye on things coming out of Google Labs⁸⁰, 37signals, and Adaptive Path.

To Sum Up

The last year has seen an explosion in the Ajax space. While the initial examples of Ajaxified applications arose from the labs of some of Silicon Valley's most talked about companies, Ajax has spread beyond the technology elite. Ajax won't cure all that ills your application but, done right, it can significantly improve your user experience. So go forth, add a dose of Ajax to your app!

Nathaniel T. Schutta

Nathaniel T. Schutta is a senior software engineer in the Twin Cities area of Minnesota with extensive experience developing Java Enterprise Edition–based Web applications. He graduated from St. John’s University (MN)⁸¹ with a degree in Computer Science⁸² and has a master’s of science degree in software⁸³ engineering from the University of Minnesota⁸⁴. For the last several years, he has focused on user interface design. Nathaniel has contributed to corporate interface guidelines and consulted on a variety of Web-based applications. A long-time member of the Association for Computing Machinery’s⁸⁵ Computer-Human Interaction Special Interest Group⁸⁶ and a Sun-certified Web component developer⁸⁷, Nathaniel believes that if the user can’t figure out your application, then you’ve done something wrong. Along with his user interface work, Nathaniel is the cocreator of the open-source Taconite⁸⁸ framework, has contributed to two corporate Java frameworks, has developed training material, and has led several study groups. During the brief moments of warm weather found in his home state of Minnesota, he spends as much time on the golf course as his wife will tolerate. He’s currently exploring Ruby⁸⁹, Rails⁹⁰, and (after recently making the switch) Mac OS X⁹¹. Nathaniel is the coauthor of the bestselling book, Foundations of Ajax⁹².



This paper contained some excerpts from Foundations of Ajax as well as Pro Ajax with Java (to be published Spring 2006). For more information, check out ntschutta.com.

Resources

- 1 <http://adaptivepath.com/>
- 2 <http://adaptivepath.com/publications/essays/archives/000385.php>
- 3 <http://blogs.msdn.com/ie/archive/2004/08/17/216080.aspx>
- 4 <http://www.ashleyit.com/rs/main.htm>
- 5 <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rmscpt/Html/rmscpt.asp>
- 6 <http://www.ashleyit.com/blogs/brentashley/>
- 7 <http://www.ashleyit.com/rs/jsrs/test.htm>
- 8 <http://me.eae.net/archive/2005/04/02/xml-http-performance-and-caching/>
- 9 <http://blogs.msdn.com/dmassy/archive/2005/03/20/399412.aspx>
- 10 <http://www.useit.com/>
- 11 <http://www.usabilityviews.com/ajaxsucks.html>
- 12 <http://swik.net/Ajax/Ajax+Mistakes>
- 13 <http://www.joelonsoftware.com/items/2005/10/21.html>
- 14 <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- 15 <http://www.paulgraham.com/web20.html>
- 16 <http://maps.google.com/>
- 17 http://news.com.com/Mapping+a+revolution+with+mashups/2009-1025_3-5944608.html
- 18 <http://googlemapsmania.blogspot.com/>
- 19 <http://www.housingmaps.com/>
- 20 <http://www.37signals.com/>
- 21 <http://www.tadalist.com/>
- 22 <http://www.backpackit.com/>
- 23 <http://en.wikipedia.org/wiki/XMLHTTP>
- 24 <http://www.joelonsoftware.com/items/2004/12/10.html>
- 25 <http://www.google.com/webhp?complete=1&hl=en>
- 26 <http://www.xulplanet.com/references/objref/XMLHttpRequest.html>
- 27 <http://developer.apple.com/internet/webcontent/xmlhttpreq.html>
- 28 <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>
- 29 <http://blogs.msdn.com/ie/archive/2005/09/13/465338.aspx>
- 30 <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
- 31 <http://www-128.ibm.com/developerworks/library/j-mocktest.html>
- 32 <http://www.mockobjects.com/FrontPage.html>
- 33 <http://www.easymock.org/>
- 34 <http://dave.sunwheeltech.com/wordpress/2005/10/17/mocking-the-server-side/>
- 35 <http://www.edwardh.com/jsunit/>
- 36 <http://getahead.ltd.uk/ajax/single-page-design>
- 37 <http://www.informationweek.com/story/showArticle.jhtml?articleID=173403044>

38 http://ajaxian.com/archives/2005/07/ajax_is_rocket.html
39
<http://www.informationweek.com/software/showArticle.jhtml?articleID=165600304&pgno=1>
40 <http://ajaxian.com/>
41 http://ajaxian.com/archives/2005/10/pragmatic_ajax.html
42 <http://www.javaworld.com/javaworld/jw-10-2005/jw-1017-ajax.html>
43 http://ajaxian.com/archives/2005/12/quirksblog_java.html
44 http://www.quirksmode.org/blog/archives/2005/11/javascript_and_1.html
45
http://news.com.com/Grassroots+computing+languages+hit+the+big+time/2100-1007_3-5705448.html
46
http://java.sun.com/developer/technicalArticles/J2SE/Desktop/Mustang_build39.html
47 <http://www.mozilla.org/rhino/>
48 <http://groovy.codehaus.org/>
49 <http://www.jython.org/>
50 <http://jruby.sourceforge.net/>
51 <http://www.mozilla.org/projects/venkman/>
52 <http://today.java.net/pub/a/today/2005/12/13/log4ajax.html>
53 <http://gleepglop.com/javascripts/logger/>
54 <http://log4js.sourceforge.net/>
55 <http://www.mozilla.org/projects/inspector/>
56 <http://users.skynet.be/mgueury/mozilla/>
57 <http://checky.sourceforge.net/extension.html>
58 <http://www.agiledata.org/essays/tdd.html>
59 <http://www.edwardh.com/jsunit/>
60 <http://www.thoughtworks.com/index.html>
61 <http://www.openqa.org/selenium/>
62 <http://www-128.ibm.com/developerworks/java/library/wa-selenium-ajax/index.html?ca=drs>
63 http://ajaxpatterns.org/Ajax_Frameworks
64 http://ajaxpatterns.org/wiki/index.php?title=Main_Page
65 <http://prototype.conio.net/>
66 <http://script.aculo.us/>
67 <http://taconite.sourceforge.net/>
68 <http://dojotoolkit.org/>
69 <https://dwr.dev.java.net/>
70 <http://www.javaworld.com/javaworld/javatips/jw-javatip67.html>
71 <http://www.netflix.com/>
72 <http://www.netflix.com/BrowseSelection?Inkctr=nmhbs>
73 <http://ajaxian.com/archives/showcase-apple-store>

74 <http://ntschutta.com/jat/2005/09/24/foundations-of-ajax/>
75 <http://www.ntschtutta.com/jat/>
76 <http://ajaxian.com/>
77 http://ajaxpatterns.org/Main_Page
78 <http://www.ajaxmatters.com/r/welcome>
79 <http://www.ajaxblog.com/>
80 <http://labs.google.com/>
81 <http://csbsju.edu/>
82 <http://www.csbsju.edu/computerscience/>
83 <http://www.msse.umn.edu/>
84 <http://www1.umn.edu/twincities/index.php>
85 <http://www.acm.org/>
86 <http://www.sigchi.org/>
87 http://www.sun.com/training/certification/java/java_web.html
88 <http://taconite.sourceforge.net/>
89 <http://www.rubycentral.com/>
90 <http://www.rubyonrails.org/>
91 <http://www.apple.com/macosex/>
92 http://www.amazon.com/gp/product/1590595823/qid=1127567332/sr=8-1/ref=pd_bbs_1/102-2209429-6756144?n=507846&s=books&v=glance