Mastering Ajax, Part 1: Introduction to Ajax

Understanding Ajax, a productive approach to building Web sites, and how it works

Skill Level: Introductory

Brett McLaughlin (brett@newInstance.com)
Author and Editor
O'Reilly Media Inc.

06 Dec 2005

Ajax, which consists of HTML, JavaScript™ technology, DHTML, and DOM, is an outstanding approach that helps you transform clunky Web interfaces into interactive Ajax applications. The author, an Ajax expert, demonstrates how these technologies work together -- from an overview to a detailed look -- to make extremely efficient Web development an easy reality. He also unveils the central concepts of Ajax, including the XMLHttpRequest object.

Five years ago, if you didn't know XML, you were the ugly duckling whom nobody talked to. Eighteen months ago, Ruby came into the limelight and programmers who didn't know what was going on with Ruby weren't welcome at the water cooler. Today, if you want to get into the latest technology rage, Ajax is where it's at.

However, Ajax is far more than *just* a fad; it's a powerful approach to building Web sites and it's not nearly as hard to learn as an entire new language.

Before I dig into what Ajax is, though, let's spend just a few moments understanding what Ajax *does*. When you write an application today, you have two basic choices:

- Desktop applications
- Web applications

These are both familiar; desktop applications usually come on a CD (or sometimes are downloaded from a Web site) and install completely on your computer. They

might use the Internet to download updates, but the code that runs these applications resides on your desktop. Web applications -- and there's no surprise here -- run on a Web server somewhere and you access the application with your Web browser.

More important than where the code for these applications runs, though, is how the applications behave and how you interact with them. Desktop applications are usually pretty fast (they're running on your computer; you're not waiting on an Internet connection), have great user interfaces (usually interacting with your operating system), and are incredibly dynamic. You can click, point, type, pull up menus and sub-menus, and cruise around, with almost no waiting around.

On the other hand, Web applications are usually up-to-the-second current and they provide services you could never get on your desktop (think about Amazon.com and eBay). However, with the power of the Web comes waiting -- waiting for a server to respond, waiting for a screen to refresh, waiting for a request to come back and generate a new page.

Obviously this is a bit of an oversimplification, but you get the basic idea. As you might already be suspecting, Ajax attempts to bridge the gap between the functionality and interactivity of a desktop application and the always-updated Web application. You can use dynamic user interfaces and fancier controls like you'd find on a desktop application, but it's available to you on a Web application.

So what are you waiting for? Start looking at Ajax and how to turn your clunky Web interfaces into responsive Ajax applications.

Old technology, new tricks

When it comes to Ajax, the reality is that it involves a lot of technologies -- to get beyond the basics, you need to drill down into several different technologies (which is why I'll spend the first several articles in this series breaking apart each one of them). The good news is that you might already know a decent bit about many of these technologies -- better yet, most of these individual technologies are easy to learn -- certainly not as difficult as an entire programming language like Java or Ruby.

Ajax defined
By the way, Ajax is shorthand for Asynchronous JavaScript and XML (and DHTML, and so on). The phrase was coined by Jesse James Garrett of Adaptive Path (see the Resources section) and is, according to Jesse, *not* meant to be an acronym.

Here are the basic technologies involved in Ajax applications:

- HTML is used to build Web forms and identify fields for use in the rest of your application.
- JavaScript code is the core code running Ajax applications and it helps facilitate communication with server applications.
- DHTML, or Dynamic HTML, helps you update your forms dynamically. You'll use `div`, `span`, and other dynamic HTML elements to mark up your HTML.
- DOM, the Document Object Model, will be used (through JavaScript code) to work with both the structure of your HTML and (in some cases) XML returned from the server.

Let's break these down and get a better idea of what each does. I'll delve into each of these more in future articles; for now focus on becoming familiar with these components and technologies. The more familiar you are with this code, the easier it will be to move from casual knowledge about these technologies to mastering each (and really blowing the doors off of your Web application development).

The XMLHttpRequest object

The first object you want to understand is probably the one that's newest to you; it's called `XMLHttpRequest`. This is a JavaScript object and is created as simply as shown in Listing 1.

Listing 1. Create a new XMLHttpRequest object

```
<script language="javascript" type="text/javascript">
var xmlHttp = new XMLHttpRequest();
</script>
```

I'll talk more about this object in the next article, but for now realize that this is the object that handles all your server communication. Before you go forward, stop and think about that -- it's the *JavaScript* technology through the `XMLHttpRequest` object that talks to the server. That's not the normal application flow and it's where Ajax gets much of its magic.

In a normal Web application, users fill out form fields and click a *Submit* button. Then, the entire form is sent to the server, the server passes on processing to a script (usually PHP or Java or maybe a CGI process or something similar), and when the script is done, it sends back a completely new page. That page might be HTML with a new form with some data filled in or it might be a confirmation or perhaps a page with certain options selected based on data entered in the original form. Of course, while the script or program on the server is processing and returning a new form, users have to wait. Their screen will go blank and then be redrawn as data comes back from the server. This is where low interactivity comes

into play -- users don't get instant feedback and they certainly don't feel like they're working on a desktop application.

Ajax essentially puts JavaScript technology and the `XMLHttpRequest` object *between* your Web form and the server. When users fill out forms, that data is sent to some JavaScript code and *not* directly to the server. Instead, the JavaScript code grabs the form data and sends a request to the server. While this is happening, the form on the users screen doesn't flash, blink, disappear, or stall. In other words, the JavaScript code sends the request behind the scenes; the user doesn't even realize that the request is being made. Even better, the request is sent asynchronously, which means that your JavaScript code (and the user) doesn't wait around on the server to respond. So users can continue entering data, scrolling around, and using the application.

Then, the server sends data back to your JavaScript code (still standing in for the Web form) which decides what to do with that data. It can update form fields on the fly, giving that immediate feeling to your application -- users are getting new data without their form being submitted or refreshed. The JavaScript code could even get the data, perform some calculations, and send another request, all without user intervention! This is the power of `XMLHttpRequest`. It can talk back and forth with a server all it wants, without the user ever knowing about what's really going on. The result is a dynamic, responsive, highly-interactive experience like a desktop application, but with all the power of the Internet behind it.

Adding in some JavaScript

Once you get a handle on `XMLHttpRequest`, the rest of your JavaScript code turns out to be pretty mundane. In fact, you'll use JavaScript code for just a few basic tasks:

- Get form data: JavaScript code makes it simple to pull data out of your HTML form and send it to the server.
- Change values on the form: It's also simple to update a form, from setting field values to replacing images on the fly.
- Parse HTML and XML: You'll use JavaScript code to manipulate the DOM (see the next section) and to work with the structure of your HTML form and any XML data that the server returns.

For those first two items, you want to be very familiar with the `getElementById()` method as shown in Listing 2.

Listing 2. Grab and set field values with JavaScript code

```
// Get the value of the "phone" field and stuff it in a variable called phone
var phone = document.getElementById("phone").value;
```

```
// Set some values on a form using an array called response
document.getElementById("order").value = response[0];
document.getElementById("address").value = response[1];
```

There's nothing particularly remarkable here and that's good! You should start to realize that there's nothing tremendously complicated about this. Once you master `XMLHttpRequest`, much of the rest of your Ajax application will be simple JavaScript code like that shown in Listing 2, mixed in with a bit of clever HTML. Then, every once in a while, there's a little DOM work...so let's look at that.

Finishing off with the DOM

Last but not least, there's the DOM, the Document Object Model. For some of you, hearing about the DOM is going to be a little intimidating -- it's not often used by HTML designers and is even somewhat unusual for JavaScript coders unless you're really into some high-end programming tasks. Where you *will* find the DOM in use a lot is in heavy-duty Java and C/C++ programs; in fact, that's probably where the DOM got a bit of its reputation for being difficult or hard to learn.

Fortunately, using the DOM in JavaScript technology is easy, and is mostly intuitive. At this point, I'd normally show you how to use the DOM or at least give you a few code examples, but even that would be misleading. You see, you can get pretty far into Ajax without having to mess with the DOM and that's the path I'm going to show you. I'll come back to the DOM in a future article, but for now, just know that it's out there. When you start to send XML back and forth between your JavaScript code and the server and really change the HTML form, you'll dig back into DOM. For now, it's easy to get some effective Ajax going without it, so put this on the back-burner for now.

Getting a Request object

With a basic overview under your belt, you're ready to look at a few specifics. Since `XMLHttpRequest` is central to Ajax applications -- and probably new to many of you -- I'll start there. As you saw in Listing 1, it should be pretty easy to create this object and use it, right? Wait a minute.

Remember those pesky browser wars from a few years back and how nothing worked the same across browsers? Well, believe it or not, those wars are still going on albeit on a much smaller scale. And, surprise: `XMLHttpRequest` is one of the victims of this war. So you'll need to do a few different things to get an `XMLHttpRequest` object going. I'll take your through it step by step.

Working with Microsoft browsers

Microsoft's browser, Internet Explorer, uses the MSXML parser for handling XML

(you can find out more about MSXML in Resources). So when you write Ajax applications that need to work on Internet Explorer, you need to create the object in a particular way.

However, it's not that easy. MSXML actually has two different versions floating around depending on the version of JavaScript technology installed in Internet Explorer, so you've got to write code that handles both cases. Look at Listing 3 for the code that you need to create an `XMLHttpRequest` on Microsoft browsers.

Listing 3. Create an XMLHttpRequest object on Microsoft browsers

```
var xmlHttp = false;
try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e2) {
    xmlHttp = false;
  }
}
```

All of this won't make exact sense yet, but that's OK. You'll dig into JavaScript programming, error handling, conditional compilation, and more before this series is finished. For now, you want to get two core lines into your head:

```
xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
```

and

```
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");.
```

In a nutshell, this code tries to create the object using one version of MSXML; if that fails, it then creates the object using the other version. Nice, huh? If neither of these work, the `xmlHttp` variable is set to false, to tell your code know that something hasn't worked. If that's the case, you've probably got a non-Microsoft browser and need to use different code to do the job.

Dealing with Mozilla and non-Microsoft browsers

If Internet Explorer isn't your browser of choice or you write code for non-Microsoft browsers, then you need different code. In fact, this is the really simple line of code you saw back in Listing 1:

```
var xmlHttp = new XMLHttpRequest object;.
```

This much simpler line creates an `XMLHttpRequest` object in Mozilla, Firefox, Safari, Opera, and pretty much every other non-Microsoft browser that supports Ajax in any form or fashion.

Putting it together

The key is to support *all* browsers. Who wants to write an application that works just on Internet Explorer or an application that works just on non-Microsoft browsers? Worse yet, do you want to write your application twice? Of course not! So your code combines support for both Internet Explorerand non-Microsoft browsers. Listing 4 shows the code to do just that.

Listing 4. Create an XMLHttpRequest object the multi-browser way

```
/* Create a new XMLHttpRequest object to talk to the Web server */
var xmlHttp = false;
/*@cc_on @*/
/*@if (@_jscript_version >= 5)
try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e2) {
    xmlHttp = false;
  }
}
@end @*/

if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
  xmlHttp = new XMLHttpRequest();
}
```

For now, ignore the commenting and weird tags like `@cc_on`; those are special JavaScript compiler commands that you'll explore in depth in my next article, which will focus exclusively on `XMLHttpRequest`. The core of this code breaks down into three steps:

1. Create a variable, `xmlHttp`, to reference the `XMLHttpRequest` object that you will create.
2. Try and create the object in Microsoft browsers:
   - Try and create the object using the `Msxml2.XMLHTTP` object.
   - If that fails, try and create the object using the `Microsoft.XMLHTTP` object.
3. If `xmlHttp` still isn't set up, create the object in a non-Microsoft way.

At the end of this process, `xmlHttp` should reference a valid `XMLHttpRequest` object, no matter what browser your users run.

A word on security

What about security? Today's browsers offer users the ability to crank their security levels up, to turn off JavaScript technology, and disable any number of options in their browser. In these cases, your code probably won't work under any circumstances. For these situations, you'll have to handle problems gracefully -- that's at least one article in itself, one I will tackle later (it's going to be a long series, isn't it? Don't worry; you'll master all of this before you're through). For now, you're writing robust, but not perfect, code, which is great for getting a handle on Ajax. You'll come back to the finer details.

Request/Response in an Ajax world

So you now understand Ajax and have a basic idea about the `XMLHttpRequest` object and how to create it. If you've read closely, you even realize that it's the JavaScript technology that talks to any Web application on the server rather than your HTML form being submitted to that application directly.

What's the missing piece? How to actually use `XMLHttpRequest`. Since this is critical code that you'll use in some form in *every* Ajax application you write, take a quick tour through what a basic request/response model with Ajax looks like.

Making a request

You have your shiny new `XMLHttpRequest` object; now take it for a spin. First, you need a JavaScript method that your Web page can call (like when a user types in text or selects an option from a menu). Then, you'll follow the same basic outline in almost all of your Ajax applications:

1. Get whatever data you need from the Web form.
2. Build the URL to connect to.
3. Open a connection to the server.
4. Set up a function for the server to run when it's done.
5. Send the request.

Listing 5 is a sample of an Ajax method that does these very things, in this order:

Listing 5. Make a request with Ajax

```
function callServer() {
  // Get the city and state from the web form
  var city = document.getElementById("city").value;
  var state = document.getElementById("state").value;
  // Only go on if there are values for both fields
```

```
    if ((city == null) || (city == "")) return;
    if ((state == null) || (state == "")) return;

    // Build the URL to connect to
    var url = "/scripts/getZipCode.php?city=" + escape(city) + "&state=" + escape(state);

    // Open a connection to the server
    xmlHttp.open("GET", url, true);

    // Setup a function for the server to run when it's done
    xmlHttp.onreadystatechange = updatePage;

    // Send the request
    xmlHttp.send(null);
}
```

A lot of this is self-explanatory. The first bit of the code uses basic JavaScript code to grab the values of a few form fields. Then the code sets up a PHP script as the destination to connect to. Notice how the URL of the script is specified and then the city and state (from the form) are appended to this using simple GET parameters.

Next, a connection is opened; here's the first place you see `XMLHttpRequest` in action again. The method of connection is indicated (GET), as well as the URL to connect to. The final parameter, when set to `true`, requests an asynchronous connection (thus making this Ajax). If you used `false`, the code would wait around on the server when the request was made and not continue until a response was received. By setting this to `true`, your users can still use the form (and even call other JavaScript methods) while the server is processing this request in the background.

The `onreadystatechange` property of `xmlHttp` (remember, that's your instance of the `XMLHttpRequest` object) allows you to tell the server what to do when it *does* finish running (which could be in five minutes or five hours). Since the code isn't going to wait around for the server, you'll need to let the server know what to do so you can respond to it. In this case, a specific method -- called `updatePage()` -- will be triggered when the server is finished processing your request.

Finally, `send()` is called with a value of `null`. Since you've added the data to send to the server (the city and state) in the request URL, you don't need to send anything in the request. So this fires off your request and the server can do what you asked it to do.

If you don't get anything else out of this, notice how straightforward and simple this is! Other than getting the asynchronous nature of Ajax into your head, this is relatively simple stuff. You'll appreciate how it frees you up to concentrate on cool applications and interfaces rather than complicated HTTP request/response code.

The code in Listing 5 is about as easy as it gets. The data is simple text and can be included as part of the request URL. GET sends the request rather than the more complicated POST. There's no XML or content headers to add, no data to send in the body of the request -- this is Ajax Utopia, in other words.

Have no fear; things will become more complicated as this series progresses. You'll learn how to send POST requests, how to set request headers and content types, how to encode XML in your message, how to add security to your request -- the list is pretty long! Don't worry about the hard stuff for now; get your head around the basics, and you'll soon build up a whole arsenal of Ajax tools.

Handling the response

Now you need to actually deal with the server's response. You really only need to know two things at this point:

- Don't do anything until the `xmlHttp.readyState` property is equal to 4.
- The server will stuff it's response into the `xmlHttp.responseText` property.

The first of these -- ready states -- is going to take up the bulk of the next article; you'll learn more about the stages of an HTTP request than you ever wanted to know. For now, if you simply check for a certain value (4), things will work (and you'll have something to look forward to in the next article). The second item -- using the `xmlHttp.responseText` property to get the server's response -- is easy. Listing 6 shows an example of a method that the server can call based on the values sent in Listing 5.

Listing 6. Handle the server's response

```
function updatePage() {
  if (xmlHttp.readyState == 4) {
    var response = xmlHttp.responseText;
    document.getElementById("zipCode").value = response;
  }
}
```

Again, this code isn't so difficult or complicated. It waits for the server to call it with the right ready state and then uses the value that the server returns (in this case, the ZIP code for the user-entered city and state) to set the value of another form field. The result is that the `zipCode` field suddenly appears with the ZIP code -- but the user *never had to click a button!*. That's the desktop application feel I talked about earlier. Responsiveness, a dynamic feel, and more, all with a little Ajax code.

Observant readers might notice that the `zipCode` field is a normal text field. Once the server returns the ZIP code and the `updatePage()` method sets the value of that field with the city/state ZIP code, users can override the value. That's intentional for two reasons: To keep things in the example simple and to show you that sometimes you *want* users to be able to override what a server says. Keep both in mind; they're important in good user-interface design.

Hooking in the Web form

So what's left? Actually, not much. You have a JavaScript method that grabs information that the user put into a form, sends it to the server, provides another JavaScript method to listen for and handle a response, and even sets the value of a field when that response comes back. All that's really left is to *call* that first JavaScript method and start the whole process. You could obviously add a button to your HTML form, but that's pretty 2001, don't you think? Take advantage of JavaScript technology like in Listing 7.

Listing 7. Kick off an Ajax process

```
<form>
 <p>City: <input type="text" name="city" id="city" size="25"
       onChange="callServer();" /></p>
 <p>State: <input type="text" name="state" id="state" size="25"
       onChange="callServer();" /></p>
 <p>Zip Code: <input type="text" name="zipCode" id="zipCode" size="5" /></p>
</form>
```

If this feels like yet one more piece of fairly routine code, then you're right -- it is! When a user puts in a new value for either the city or state field, the `callServer()` method fires off and the Ajax fun begins. Starting to feel like you've got a handle on things? Good; that's the idea!

In conclusion

At this point, you're probably not ready to go out and write your first Ajax application -- at least, not unless you're willing to do some real digging in the Resources section. However, you can start to get the basic idea of how these applications work and a basic understanding of the `XMLHttpRequest` object. In the articles to come, you'll learn to master this object, how to handle JavaScript-to-server communication, how to work with HTML forms, and even get a handle on the DOM.

For now, though, spend some time thinking about just how powerful Ajax applications can be. Imagine a Web form that responds to you not just when you click a button, but when you type into a field, when you select an option from a combo box...even when you drag your mouse around the screen. Think about exactly what *asynchronous* means; think about JavaScript code running and *not waiting* on the server to respond to its requests. What sorts of problems can you run into? What areas do you watch out for? And how will the design of your forms change to account for this new approach in programming?

If you spend some real time with these issues, you'll be better served than just having some code you can cut-and-paste and throw into an application that you

really don't understand. In the next article, you'll put these ideas into practice and I'll give you the details on the code you need to really make applications like this work. So, until then, enjoy the possibilities of Ajax.

Resources

Learn

- "Use Ajax with WebSphere Portal" (developerWorks, June 2006) to improve portal performance, create a cleaner portal application architecture, and -- most important -- give your users a much more responsive portal.
- Adaptive Path is one of the companies on the leading edge of user interface design; you can learn a ton about Ajax by perusing their pages.
- If you're curious about where the term Ajax came from, check out Jesse James Garrett and his excellent articles (like this one) on Ajax.
- You can get a head start on the next article in this series, focusing on the `XMLHttpRequest` object, by checking out this excellent article on the `XMLHttpRequest` object.
- If you use Internet Explorer, you can get the scoop at the Microsoft Developer Network's XML Developer Center.
- Ajax for Java developers: Build dynamic Java applications (developerWorks, September 2005) introduces a groundbreaking approach to creating dynamic Web application experiences that solve the page-reload dilemma.
- Ajax for Java developers: Java object serialization for Ajax (developerWorks, October 2005) shows you five ways to serialize data in Ajax applications.
- Using Ajax with PHP and Sajax (developerWorks, October 2005) is a tutorial for those interested in developing rich Web applications that dynamically update content using Ajax and PHP.
- Call SOAP Web services with AJAX, Part 1: Build the Web services client (developerWorks, October 2005) shows how to implement a Web browser-based SOAP Web services client using the Ajax design pattern.
- XML Matters: Beyond the DOM (developerWorks, May 2005) details the Document Object Model as a method to build dynamic Web applications.
- Build apps with Asynchronous JavaScript with XML, or AJAX (developerWorks, November 2005) demonstrates how to construct real-time-validation-enabled Web applications with AJAX.
- Ajax for Java developers: Ajax with Direct Web Remoting (developerWorks, November 2005) demonstrates how to automate the heavy-lifting of AJAX.
- The OSA Foundation has a wiki that surveys AJAX/JavaScript libraries.
- XUL Planet's object reference section details XMLHttpRequest (not to mention all kinds of other XML objects, as well as DOM, CSS, HTML, Web Service, and Windows and Navigation objects.

- See one of the outstanding Ajax applications online at Flickr.com.
- GMail, from Google, is another great example of Ajax revolutionizing Web applications.
- *Head Rush Ajax* (O'Reilly Media, Inc., February 2006) takes the ideas outlined in this article and series and brings them (and a lot more) to you in the innovative and award-winning Head First format.
- *JavaScript: The Definitive Guide* , 4th Edition (O'Reilly Media, Inc., November 2001) is a great resource for the JavaScript language and working with dynamic Web pages.
- The developerWorks Web Architecture zone specializes in articles covering various Web-based solutions.

Discuss

- Participate in the discussion forum for this content.
- Ajax.NET Professional is a great blog for all things Ajax.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Brett McLaughlin
Brett McLaughlin has worked in computers since the Logo days.(Remember the little triangle?) In recent years, he's become one of the most well-known authors and programmers in the Java and XML communities. He's worked for Nextel Communications, implementing complex enterprise systems; at Lutris Technologies, actually writing application servers; and most recently at O'Reilly Media, Inc., where he continues to write and edit books that matter. Brett's upcoming book, *Head Rush Ajax* , brings the award-winning and innovative Head First approach to Ajax, along with bestselling co-authors, Eric and Beth Freeman. His last book, *Java 1.5 Tiger: A Developer's Notebook* , was the first book available on the newest version of Java technology and his classic *Java and XML* remains one of the definitive works on using XML technologies in the Java language.