

# Considering Ajax, Part 1: Cut through the hype

Learn when and how to implement this new technology

Level: Introductory

Chris Laffra ([laffrac@us.ibm.com](mailto:laffrac@us.ibm.com)), Performance Engineering Team Lead, IBM

09 May 2006

Lately, interest in Ajax (Asynchronous JavaScript and XML) is high. Various Ajax applications provide a more interactive and rich client experience than traditional Web pages. Using Ajax, you can deploy new and innovative aggregation and presentation techniques in an unprecedented fashion. Inspired by Alex Bosworth's list of Ajax mistakes, Chris Laffra has compiled a set of discussion points for every developer to consider before using Ajax techniques for a Web site that he outlines in this two-part series. Some points are potential problem areas; most highlight Ajax's great potential.

The hottest topic these days on venture capital (VC) calendars and developer blogs is Ajax. Major ingredients of Ajax have been around for over a decade, so there really isn't much new to Ajax. However, the name *Ajax*, which stands for *Asynchronous JavaScript and XML*, is pretty new; it was introduced in 2005 by Jesse James Garrett of Adaptive Path. Leaning heavily on Dynamic HTML (DHTML), Ajax tries to avoid round trips to the server. When an Ajax application updates some information, it does not replace the entire page in the Web browser. Instead, JavaScript code sends an XML request to the server, and at some later point replaces a selected subset of the DOM to update the current page.

Before the advent of Ajax, DHTML was usually restricted to the implementation of menus or tables of contents, limited forms of animation, and tabbed folders. To make a DHTML Web site portable between various browsers is difficult. This created a market for companies like Bindows™ that offer comprehensive abstractions for developing interesting rich clients; the online version of the Eclipse™ FAQs is a good example (see [Resources](#) for a link to this and other sites.)

Arguably, the most influential Ajax application so far is Google™ Maps. It has generated a whole new computer science discipline. (A search at Google for "Hacking Google Maps" returns over 2 million hits; go figure.) Nice examples of Google Map hacks include transparent maps, Bus Monster, and HousingMaps.com. Personally, I am most intrigued by Google's personalization features, where you can use DHTML to drag and position existing and new content on a personalized home page. There is an inkling of custom mashup as users can perform any Google search and add a selected host to their home page.

If you're considering building an Ajax application of your own, you will inevitably need to consider the discussion points in this series. You'll learn about both the potential and the pitfalls inherent to this new technology. Note that this article is not for Ajax beginners; I don't really offer a tutorial in using `XmlHttpRequest`, for instance. Rather, it's for those who are comfortable with the technical basics but are looking for some big-picture guidance on implementing Web sites using Ajax features. [Resources](#) has some pointers to introductory material on Ajax.

## If I had a hammer...

"When we were young, we were told that 'Everybody else is doing it' was a really stupid reason

### Alphabet soup

Every new technology introduces its own lingo, and Ajax is no exception. Here are few abbreviations relevant to the Ajax phenomenon:

- **Ajax:** Asynchronous JavaScript with XML
- **XML:** Extensible Markup Language
- **RIA:** Rich internet application
- **RSS:** Really Simple Syndication
- **PS:** Paradigm shift
- **TP:** Tipping point
- **VC:** Venture capital

to do something. Now it's the standard reason for picking a particular software package." -- Barry Gehm

When you make decisions, one strong cognitive bias is the *bandwagon effect*, a well-known psychological phenomenon (see [Resources](#) for a description). The idea is that humans often do or think things just because many others do the same. Now, after convincing yourself that Ajax is something you *have* to use, be careful not to find artificial excuses to use it. To some people, Ajax is the latest proverbial hammer, and they will use any excuse to find themselves some suitable nails. Remember that Ajax is a tool, not a toy or purpose in itself.

With each new technology that is introduced, people like to experiment and test the tipping point. For instance, when color monitors first came out (most Web developers are too young to remember this), a deluge of applications changed their fonts and colors as much as possible, just because they could. No user really wants a mouse trailer, yet a large number of Web designers seem to think that they have to add them to their sites. If you're not careful, the same phenomenon will happen with Ajax.

Ajax is also fueling a venture capital frenzy. The investment climate for technology startups seems to have become friendly again, and one of the magic keywords these days is Ajax. Collaboration-based startup Zimbra™, a heavy Ajax user, has already secured \$16 million. Other startups are scrambling to claim that they are Ajax-based as a result, to sell more products or receive more attention from investors.

If you find yourself wanting to use Ajax, make a balanced decision about where exactly you think it makes sense, and where it does not. Hopefully, the two installments in this series will give you proper ammunition.

---

## Frameworkitis

Browser implementations vary across a few key areas:

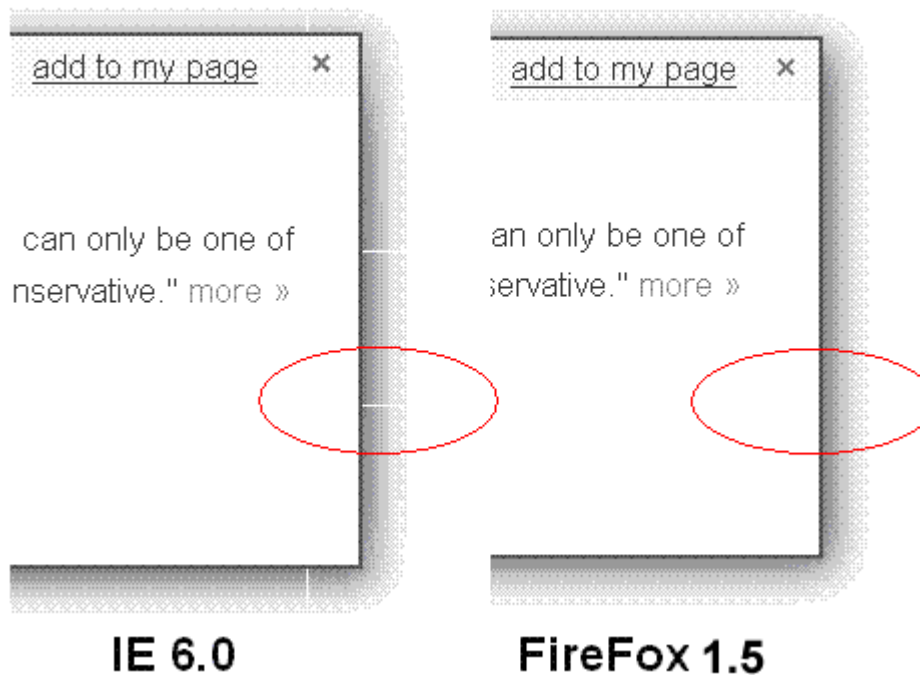
- Event registration and event handling
- HTTP request object implementation
- Document Object Model (DOM) API

For a long time, these incompatibilities have impeded adoption of heavy JavaScript usage in the browser. It is just too tedious to write browser test statements everywhere to enable the user to, say, drag a `div` across a layout. Google has paved the way by showing that with enough patience, you can develop a suitable framework that is truly cross-platform and that supports the majority of the browsers (though not all of them; Google's Ajax applications currently do not support Opera, for instance).

Inspired by the success of Google, dozens of Ajax frameworks have sprouted. The most notable entry is Atlas from Microsoft® -- notable not because it is particularly well-designed or aesthetically pleasing, but because it is specifically positioned as a browser-neutral Ajax framework. It modernizes the JavaScript language by defining a richer base class library, adding inheritance and interfaces, and making composition easier by defining namespaces. Good demonstrations of Atlas in action include Virtual Earth, Outlook Web access, and Start.com (see [Resources](#) for links to all three). The latter is illustrated in [Figure 1](#), which shows real innovation: Atlas is one of the first Microsoft applications that works better in Firefox® than in Internet Explorer®.

**Figure 1. Atlas browser differences (notice that the shadow images align better in Firefox than in Internet Explorer)**

## Atlas browser differences



Resist the temptation to write your own framework. This is painful and tedious stuff.

For a comprehensive list of available Ajax frameworks, see [Resources](#).

## Amnesia

Web browsing relies heavily on following hyperlinks, building a history of visited pages, and allowing users to retreat to previously visited pages. The most prominent and commonly used button in a Web browser is the **Back** button (see [Figure 2](#)), and there is a good reason for this.

**Figure 2. The Back button is one of the most prominent buttons in a browser**



But with the advent of Ajax techniques, developers are more and more tempted to design so-called *single-page* HTML applications: amnesic pages without any history. Many Ajax applications bootstrap themselves and never change their URL. If such a site does not support the **Back**, **Stop**, and **Refresh** buttons, as well as bookmarks, users might follow tedious navigation paths to return to their intended state and have difficulty sharing a given site with other people.

For a user who searches for various addresses in succession, Google Maps until recently made no attempt to remember history; the **Back** button was hopelessly broken. Thanks to the continuous release model that is

natural for Web applications, at some point in time Google quietly fixed this problem: try it out now and you'll find that the **Back** and **Forward** buttons actually do work. However, it's still hard to bookmark a random search result with Google Maps. Even though it supports navigation history, Google Maps does not update the URL in the browser's address section. Instead, it offers a "Link to this page" macro that will change the document's base URL so that a bookmark can be created. Wikis have the same problem, and typically include a "Permanent link" button.

In conclusion of this point, Ajax applications should pay particular attention to properly maintaining the browser's history. The basic rule is: if you allow someone to click on a link, you have to add an object to the browser history so the **Back** button will work as expected. As a bonus, add support for bookmarkability through permanent links.

---

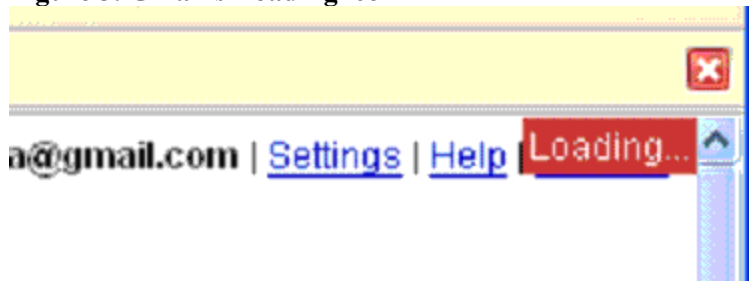
## Feedback

When a user triggers certain Ajax actions, your application must give an immediate visual cue to provide feedback to users so that they know that they actually clicked the mouse at the proper spot. Due to the unreliable nature of HTTP requests over the Internet, give special care to meeting general, well-known user interface (UI) response times:

- Provide feedback instantly, or within 0.1 second.
- Use at most four seconds for an operation. If necessary, redesign your application. Make sure that your Ajax operations are quick and provide immediate responses. Break up long-running operations into smaller chunks to meet this goal. This is something search engines do by returning, say, 20 results per page, rather than 1 million.
- Use progress bars when operations take more than four seconds. Note however that due to Ajax's asynchronous nature, showing reliable progress in Ajax applications is very hard.

If possible, when pauses are unavoidable, use something like the **Loading** icon in Gmail™, illustrated in [Figure 3](#).

**Figure 3. Gmail's Loading icon**



Here is a simple recipe: Start a timer to show a notification in 100 milliseconds. If the asynchronous result you're waiting for comes back, make the DOM changes necessary to display it, and then remember that you processed the result. If the 100 milliseconds timer goes off after everything has been updated, ignore it. Otherwise, your state will indicate that a request is still pending, and you should draw the appropriate feedback on-screen. Abstract and hide all this logic in a `XmlHttpRequest` service, of course, and support multiple concurrent requests.

Give feedback as locally as possible. The feedback given by Gmail is actually not a good model in general, as there may be a large distance between the mouse cursor and the visual feedback. In such cases, it might be smart to use a wait cursor also. Alternatively, when expanding a node in a tree, it is best to first show a child node that says something like "Getting quotes...", and then replace the temporary node with the final result when the query comes back from the server.

---

## Cold turkey

Last month, I was on my way to visit a friend. It was dark and I got lost, and I tried to find his address in my computer. His coordinates were included in the e-mail he sent me that day. Unluckily for me, that e-mail message was sent to my GMail account, and, being disconnected from the Internet, I was left with quite a negative experience. In one split second, all the benefits of zero-install, a cool UI, labels, free targeted advertisements, an extremely useful search engine, and platform independence were annihilated when I could not find my friend's address because I was out of range of a Wi-Fi hotspot.

Being disconnected from a network is less of an issue for enterprise applications, or for applications such as Google search, where there is no off-line alternative, really. However, for applications that replace existing rich clients, being disconnected might result in an experience that's much worse than using than the old app.

Gmail has POP support, so users can prepare ahead of time and decide to use Microsoft Outlook® or some other POP client to read their mail, even when offline. The transition is not painless, and requires proactive user collaboration. The main problem is that Ajax includes no obvious built-in solution to support offline browsing.

---

## Making me think

The major reason for the success of the Web is the predictability and simplicity of its UI model. Basically, anyone can move a mouse, click on a link, move a scrollbar, and hit the **Back** button. With the growing popularity of Ajax, the risk is very real that developers will go overboard and essentially make everything clickable and change the UI in an unexpected and asynchronous manner. The last thing you want to do is force your user to think. For those who are interested in the usability of Web applications, Steve Krug's *Don't Make Me Think* (see [Resources](#)) is a must-read. It does not cover Ajax specifically, but teaches important lessons about how important good design is for a Web site.

Ajax is definitely the domain of programmers, and they are notoriously bad at designing UIs. No matter how hard programmers try, their UIs look like they are designed by geeks. For Ajax-based applications to become fully successful, user interface designers and graphic artists have to be empowered to design a front-end based on HTML, CSS, and Ajax, without needing help from programmers on the presentation side. The key to the solution of this problem is left up to Ajax development tools -- the next problem area.

---

## Development tools

Ajax is not a technology by itself, but rather a combination of a few key ideas. It is the actual synthesis of these ideas that makes Ajax special. With Ajax having received interest only quite recently, integrated development environment (IDE) vendors are still in catch-up mode. As a result, development tools for Ajax are fragmentary, and each focuses on only part of the larger Ajax development picture:

- **Design.** This is the domain of Adobe® Dreamweaver, easily covering more bookshelves in bookstores than any other computer-related topic.
- **Editing.** Various IDEs, such as the Eclipse Web Tools Platform (WTP) project, support syntax highlighting for HTML and JavaScript pages. Java developers will be disappointed with the limited amount of content assistance that is provided. Due to the dynamic nature of JavaScript, no type information is available.

- **Business logic.** This is code that traditionally runs on application servers, primarily accessed through Web services.
- **Debugging.** See Venkman for Firefox and Script Debugger for Internet Explorer.
- **Profiling.** See Venkman for Firefox. None known for Internet Explorer.

(You can find links to all of these tools in [Resources](#)).

The first IDE to corner the market will be the one that takes these individual focus areas and comes up with a comprehensive environment that is easy to use for graphic artists, business analysts, Java coders, and database administrators. I expect the Eclipse Ajax Tooling Framework to be able to fill that role.

---

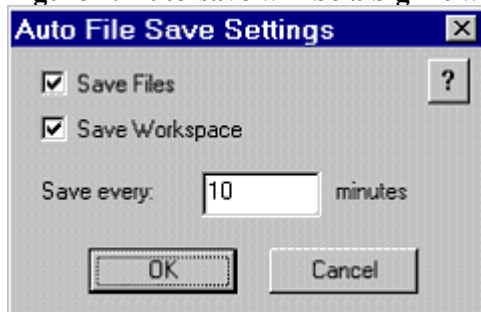
## Concurrency

The first letter in Ajax stands for *asynchronous* communication. This communication is said to be asynchronous because events do not arrive immediately in response to requests, but some time later. They may take a long time to arrive (see the discussion above on [feedback](#)); in fact, the queries might never be answered. The Web is an unreliable world, and contingency planning should be part of any Ajax application. Because multiple concurrent XML HTTP requests can be scheduled in parallel, our application essentially becomes a concurrent program -- and concurrency is hard.

What will save most developers is the fact that JavaScript interpreters are single-threaded. In other words, at any point in time either an event handler or an XML HTTP event handler is running. Callbacks never run in parallel. Therefore, complicated locking to ensure coordinated access to the DOM, for instance, isn't necessary. JavaScript does not understand the concept of threads, and, as a result, also lacks a sleep function. Everything runs in the foreground, and a sleep function could easily block the entire browser.

Ajax applications have a state that is constantly in flux. Proper management of this state is essential, and fragmenting the state into many unrelated global variables is definitely not the proper way to go. Furthermore, one has to give the user the ability to jump to a given state instantly -- if he or she wants to stop working and return to the current state a week later, for instance. Finally, in long-lasting editing sessions, users will be very grateful if you implement auto-save (as illustrated in [Figure 4](#)).

**Figure 4. Auto-save will be a big hit with your users**



In conclusion, manage state transitions in an Ajax application explicitly to avoid unhappy users, broken applications, and debugging nightmares.

---

## "Open source" (by default)

JavaScript applications run in a browser, and can be easily reengineered. By loading JavaScript files on

demand, you can fool Internet Explorer users; but other browsers, such as Firefox, will eagerly show a user the current DOM in its entirety through the context menu's **View Selection Source** option. If someone really wants to see your application's entire JavaScript source and analyze it, a simple script built with the Mozilla® Greasemonkey extension, a debugger like Venkman, or a custom Internet Explorer toolbar would do the trick.

Realize that JavaScript is as insecure as the HTML it ships with, and make sure to put the value of your application in the realms of data composition and ease of use. When publishing applications on the Internet, be careful about moving too much of your business logic into Ajax and away from the server. For intranet applications, this is less of a concern, and more logic could be pushed to the client, so as to reduce the load on company servers.

---

## Paranoia runs deep

Ajax does not change anything as far as security is concerned. The techniques underlying Ajax have been available for a long time. Hidden frames, so-called `iframes`, have been around for a decade, and Microsoft has long supported the notion of remote scripting. If there are any security risks threatening the advance of Ajax, a lot of existing Web sites suffer from them already.

Web site developers should be wary of anyone spoofing the GET and POST requests sent to them. With the availability of technologies such as Greasemonkey, users can drastically change what is being sent to the server, and how it is then represented in the browser. Ajax uses the same security mechanism as normal Web browsing, nothing more and nothing less. Whenever you send sensitive data over an `XmlHttpRequest`, use proper authentication and secure tunneling.

What Ajax will promote, of course, is the *componentization* of Web applications into much lower level chunks of functionality or services. This trend has been called *atomization*. What Web developers will have to contend with is their application turning into a collection of small components. These fragments can be approached in unexpected sequences, maliciously or not. A formalized workflow model will become more important than it has been.

---

## Tune in next time...

In this installment of *Considering Ajax*, I discussed the hype that currently surrounds this technology. You also saw that reliable frameworks are still under construction, and that you should worry about navigation history, bookmarkability, feedback, persistence, concurrency, and security.

In the next installment of this series, you will look at other Ajax topics, such as dealing with document repaints, testing, publish and subscribe models, performance, accessibility, support for older browsers, and stating our intentions. I'll also talk about interesting opportunities that Ajax offers for developing Web sites inside Web sites.

---

## Resources

### Learn

- "[Introduction to Ajax](#)," (Brett McLaughlin, developerWorks, December 2005): Get started with Ajax and see how this productive approach to building Web sites works.
- [bandwagon effect](#): Find out more about this term from Wikipedia.

- [Alex Bosworth's list of Ajax mistakes](#): Explore part of the inspiration for this series.
- The [Eclipse FAQs](#): Visit this good example of an Ajax site.
- *Don't Make Me Think*, (Steve Krug, New Riders Press, 2000): Read this excellent book on the essentials of Web design.
- [Ajax Frameworks](#): Check out this comprehensive list of Ajax frameworks -- that you really don't want to build yourself.
- [developerWorks Web Architecture zone](#): Expand your site development skills with articles and tutorials that specialize in Web technologies.
- [developerWorks technical events and webcasts](#): Stay current with jam-packed technical sessions that shorten your learning curve, and improve the quality and results of your most difficult software projects.

### Get products and technologies

- [Bindows](#): Learn more about these tools for developing efficient and flexible Rich Internet Applications.
- [Google Maps](#): Check out a successful and influential Ajax application.
- [Bus Monster](#) and [HousingMaps.com](#): Explore these Google Maps-based apps online.
- [Adobe Dreamweaver](#), the [Eclipse WTP project](#), the [Eclipse Ajax Tooling Framework](#), the [Venkman JavaScript Debugger](#), and [Microsoft Script Debugger](#): No single tool rules the Ajax development space. Most developers work on different parts of a project with a variety of tools.
- [Greasemonkey](#) is a useful tool that, among other things, can help you analyze Ajax applications running online.
- [Virtual Earth](#): Find directions, interactive maps, and satellite/aerial imagery of the United States.
- [Outlook Web Access](#): Investigate this Web cousin to Microsoft Outlook.
- [Start.com](#): Try this news gatherer that uses Ajax (based on Atlas) and is functionally compatible to personalized Google.

### Discuss

- [Participate in the discussion forum](#).
- [developerWorks blogs](#): Get involved in the developerWorks community.
- [developerWorks discussion forums](#): Join the discussion threads that interest you.

---

## About the author

Chris Laffra was born in the Netherlands and obtained his MsC at the Vrije Universiteit of Amsterdam in 1988 and a PhD at the Erasmus University of Rotterdam in 1992. At both IBM T.J. Watson Research Center

and Morgan Stanley, Chris worked on tools for user interfaces, component infrastructures, program analysis, debugging, visualization, compression, and optimization. He led the OTI Amsterdam lab for three and a half years, working on WebSphere Studio Device Developer. At IBM Canada's lab in Ottawa, he worked on the border between Java runtime environments and Eclipse (and co-authored the Official Eclipse 3.0 FAQs). Currently, Chris works as Performance Engineering Team Lead at IBM Rational to improve RAD/RSA performance. He has experimented a lot with Ajax in the past, doing things such as enhancing Google Maps to find a new home in Raleigh near a good school, and generating the [online version of the Eclipse FAQs](#).

---