

Beginning Ajax with PHP

From Novice to Professional



Lee Babin

Beginning Ajax with PHP: From Novice to Professional

Copyright © 2007 by Lee Babin

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-667-8

ISBN-10 (pbk): 1-59059-667-6

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

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Jason Gilmore

Technical Reviewer: Quentin Zervaas

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 Flores

Copy Editors: Damon Larson, Jennifer Whipple

Assistant Production Director: Kari Brooks-Copony

Production Editor: Laura Esterman

Composer: Dina Quan

Proofreader: Lori Bring

Indexer: John Collin

Artist: April Milne

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 information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Source Code/Download section.

Contents at a Glance

About the Author	ix
About the Technical Reviewer	xi
Acknowledgments	xiii
Introduction	xv
CHAPTER 1 Introducing Ajax	1
CHAPTER 2 Ajax Basics	11
CHAPTER 3 PHP and Ajax	25
CHAPTER 4 Database-Driven Ajax	49
CHAPTER 5 Forms	67
CHAPTER 6 Images	87
CHAPTER 7 A Real-World Ajax Application	101
CHAPTER 8 Ergonomic Display	123
CHAPTER 9 Web Services	135
CHAPTER 10 Spatially Enabled Web Applications	149
CHAPTER 11 Cross-Browser Issues	175
CHAPTER 12 Security	187
CHAPTER 13 Testing and Debugging	205
CHAPTER 14 The DOM	217
INDEX	235

Contents

About the Author	ix
About the Technical Reviewer	xi
Acknowledgments	xiii
Introduction	xv
CHAPTER 1 Introducing Ajax	1
From CGI to Flash to DHTML	2
Pros and Cons of Today's Web Application Environment	3
Enter Ajax	4
Ajax Requirements	8
Summary	9
CHAPTER 2 Ajax Basics	11
HTTP Request and Response Fundamentals	11
The XMLHttpRequest Object	13
XMLHttpRequest Methods	13
XMLHttpRequest Properties	15
Cross-Browser Usage	17
Sending a Request to the Server	19
Basic Ajax Example	20
Summary	24
CHAPTER 3 PHP and Ajax	25
Why PHP and Ajax?	25
Client-Driven Communication, Server-Side Processing	26
Basic Examples	26
Expanding and Contracting Content	26
Auto-Complete	32
Form Validation	41
Tool Tips	44
Summary	47

CHAPTER 4	Database-Driven Ajax	49
	Introduction to MySQL	50
	Connecting to MySQL	51
	Querying a MySQL Database	52
	MySQL Tips and Precautions	57
	Putting Ajax-Based Database Querying to Work	58
	Auto-Completing Properly	60
	Loading the Calendar	63
	Summary	65
CHAPTER 5	Forms	67
	Bringing in the Ajax: GET vs. POST	68
	Passing Values	69
	Form Validation	80
	Summary	86
CHAPTER 6	Images	87
	Uploading Images	87
	Displaying Images	91
	Loading Images	94
	Dynamic Thumbnail Generation	95
	Summary	99
CHAPTER 7	A Real-World Ajax Application	101
	The Code	102
	How It Looks	111
	How It Works	113
	Summary	122
CHAPTER 8	Ergonomic Display	123
	When to Use Ajax	124
	Back Button Issues	125
	Ajax Navigation	125
	Hiding and Showing	127
	Introduction to PEAR	128
	HTML_Table	129
	Summary	134

CHAPTER 9	Web Services	135
	Introduction to SOAP Web Services	136
	Bring in the Ajax	137
	Let's Code	137
	How the SOAP Application Works	142
	Summary	147
CHAPTER 10	Spatially Enabled Web Applications	149
	Why Is Google Maps so Popular?	149
	Where to Start	151
	How Our Mapping System Works	163
	Summary	174
CHAPTER 11	Cross-Browser Issues	175
	Ajax Portability	175
	Saving the Back Button	177
	Ajax Response Concerns	180
	Degrading JavaScript Gracefully	183
	The noscript Element	184
	Browser Upgrades	185
	Summary	185
CHAPTER 12	Security	187
	Increased Attack Surface	187
	Strategy 1: Keep Related Entry Points Within the Same Script	188
	Strategy 2: Use Standard Functions to Process and Use User Input	188
	Cross-Site Scripting	189
	Strategy 1: Remove Unwanted Tags from Input Data	191
	Strategy 2: Escape Tags When Outputting Client-Submitted Data	192
	Strategy 3: Protect Your Sessions	192
	Cross-Site Request Forgery	193
	Confirming Important Actions Using a One-Time Token	193
	Confirming Important Actions Using the User's Password	195
	GET vs. POST	195
	Accidental CSRF Attacks	195

Denial of Service	196
Strategy 1: Use Delays to Throttle Requests	197
Strategy 2: Optimize Ajax Response Data	198
Protecting Intellectual Property and Business Logic	200
Strategy 1: JavaScript Obfuscation	200
Strategy 2: Real-Time Server-Side Processing	201
Summary	204
CHAPTER 13 Testing and Debugging	205
JavaScript Error Reporting	205
Firefox Extensions	208
Web Developer Toolbar	208
The DOM Inspector	208
LiveHTTPHeaders	209
Venkman JavaScript Debugger	211
HTML Validation	212
Internet Explorer Extensions	213
Internet Explorer Developer Toolbar	214
Fiddler	215
Summary	216
CHAPTER 14 The DOM	217
Accessing DOM Elements	217
document.getElementById	217
getElementsByTagName	218
Accessing Elements Within a Form	219
Adding and Removing DOM Elements	219
Manipulating DOM Elements	221
Manipulating XML Using the DOM	222
Combining Ajax and XML with the DOM	223
How the Ajax Location Manager Works	228
Summary	233
INDEX	235

About the Author



■ **LEE BABIN** is a programmer based in Calgary, Alberta, where he owns and operates an innovative development firm duly named Code Writer. He has been developing complex web-driven applications since his graduation from DeVry University in early 2002, and has since worked on over 100 custom web sites and online applications.

Lee is married to a beautiful woman by the name of Dianne, who supports him in his rather full yet rewarding work schedule. Lee and Dianne are currently expecting their first child, and Lee cannot wait to be a father.

Lee enjoys video games, working out, martial arts, and traveling, and can usually be found working online on one of his many fun web projects.

About the Technical Reviewer

■ **QUENTIN ZERVAAS** is a web developer from Adelaide, Australia. After receiving his degree in computer science in 2001 and working for several web development firms, Quentin started his own web development and consulting business in 2004.

In addition to developing custom web applications, Quentin also runs and writes for `phpRiot()`, a web site about PHP development. The key focuses of his application development are usability, security, and extensibility.

In his spare time, Quentin plays the guitar and basketball, and hopes to publish his own book on web development in the near future.

Acknowledgments

Writing a book is never a simple process. It relies on the help and understanding of many different people to come to fruition. Writing this book was no exception to the rule; it truly could not have come together in its completed form without the understanding and assistance of a select few.

First and foremost, I would like to thank a very talented, dedicated, and highly skilled individual by the name of Quentin Zervaas. Quentin consistently volunteered his time and hard effort to ensure the absolute quality of the content found within this book. He worked tirelessly to ensure that every last snippet and concept was as polished as could possibly be. Then, during a particularly difficult period in the writing process, Quentin played a key role in ensuring the book made its way to the bookshelf. It would be a vast understatement to say that there is no way I could have completely this book without him. Thank you Quentin—your assistance during hard times is truly appreciated.

While you might suppose that a book is written and finalized by the author alone, there are always key players that help to ensure that any book is completed on schedule and of the highest quality. This book is no exception, and I would truly like to thank Jason Gilmore and Richard Dal Porto for both managing the book and ensuring that it made it through to finalization. Jason and Richard both helped immensely, and I would like to thank them very much for having the patience and understanding to see it through to the end.

I would also like to thank my loving wife, Dianne, for putting up with some insanely long hours of work and for not being upset at me despite my having no time to spend with her for months on end. She is the one who continued to support me throughout the project and I could not have finished it without her constant patience, love, support, and assurance.

Lastly, I would like to thank you, the reader. While I am sure that is something of a cliché, it truly means a lot to me that you hold this book in your hands (or are viewing it on your laptop). I suppose it goes without saying that there is no point writing something if no one reads it. I appreciate your support and I truly hope you enjoy this book and find it very useful.

Introduction

Working with technology is a funny thing in that every time you think you have it cornered . . . blam! Something pops out of nowhere that leaves you at once both bewildered and excited. Web development seems to be particularly prone to such surprises. For instance, early on, all we had to deal with was plain old HTML, which, aside from the never-ending table-wrangling, was easy enough. But soon, the simple web site began to morph into a complex web application, and accordingly, scripting languages such as PHP became requisite knowledge. Server-side development having been long since mastered, web standards such as CSS and XHTML were deemed the next link in the Web's evolutionary chain.

With the emergence of Ajax, developers once again find themselves at a crossroads. However, just as was the case with the major technological leaps of the past, there's little doubt as to which road we'll all ultimately take, because it ultimately leads to the conclusion of clicking and waiting on the Web. Ajax grants users the luxury of accessing desktop-like applications from any computer hosting a browser and Internet connection. Likewise, developers now have more reason than ever to migrate their applications to an environment that has the potential for unlimited users.

Yet despite all of Ajax's promise, many web developers readily admit being intimidated by the need to learn JavaScript (a key Ajax technology). Not to worry! I wrote this book to show PHP users how to incorporate Ajax into their web applications without necessarily getting bogged down in confusing JavaScript syntax, and I've chosen to introduce the topic by way of practical examples and real-world instruction. The material is broken down into 14 chapters, each of which is described here:

Chapter 1: "Introducing Ajax," puts this new Ajax technology into context, explaining the circumstances that led to its emergence as one of today's most talked about advancements in web development.

Chapter 2: "Ajax Basics," moves you from the why to the what, covering fundamental Ajax syntax and concepts that will arise no matter the purpose of your application.

Chapter 3: "PHP and Ajax," presents several examples explaining how the client and server sides come together to build truly compelling web applications.

Chapter 4: "Database-Driven Ajax," builds on what you learned in the previous chapter by bringing MySQL into the picture.

Chapter 5: "Forms," explains how Ajax can greatly improve the user experience by performing tasks such as seemingly real-time forms validation.

Chapter 6: "Images," shows you how to upload, manipulate, and display images the Ajax way.

Chapter 7: “A Real-World Ajax Application,” applies everything you’ve learned so far to build an Ajax-enabled photo gallery.

Chapter 8: “Ergonomic Display,” touches upon several best practices that should always be applied when building rich Internet applications.

Chapter 9: “Web Services,” shows you how to integrate Ajax with web services, allowing you to more effectively integrate content from providers such as Google and Amazon.

Chapter 10: “Spatially Enabled Web Applications,” introduces one of the Web’s showcase Ajax implementations: the Google Maps API.

Chapter 11: “Cross-Browser Issues,” discusses what to keep in mind when developing Ajax applications for the array of web browsers in widespread use today.

Chapter 12: “Security,” examines several attack vectors introduced by Ajax integration, and explains how you can avoid them.

Chapter 13: “Testing and Debugging,” introduces numerous tools that can lessen the anguish often involved in debugging JavaScript.

Chapter 14: “The DOM,” introduces the document object model, a crucial element in the simplest of Ajax-driven applications.

Contacting the Author

Lee can be contacted at lee@babinplanet.ca.



Introducing Ajax

Internet scripting technology has come along at a very brisk pace. While its roots are lodged in text-based displays (due to very limited amounts of storage space and memory), over the years it has rapidly evolved into a visual and highly functional medium. As it grows, so do the tools necessary to maintain, produce, and develop for it. As developers continue to stretch the boundaries of what they can accomplish with this rapidly advancing technology, they have begun to request increasingly robust development tools.

Indeed, to satisfy this demand, a great many tools have been created and made available to the self-proclaimed “web developer.” Languages such as HTML, PHP, ASP, and JavaScript have arisen to help the developer create and deploy his wares to the Internet. Each has evolved over the years, leaving today’s web developer with an amazingly powerful array of tools. However, while these tools grow increasingly powerful every day, several distinctions truly separate Internet applications from the more rooted desktop applications.

Of the visible distinctions, perhaps the most obvious is the page request. In order to make something happen in a web application, a call has to be made to the server. In order to do that, the page must be refreshed to retrieve the updated information from the server to the client (typically a web browser such as Firefox or Internet Explorer). This is not a browser-specific liability; rather, the HTTP request/response protocol inherent in all web browsers (see Figure 1-1) is built to function in this manner. While theoretically this works fine, developers have begun to ask for a more seamless approach so that their application response times can more closely resemble the desktop application.

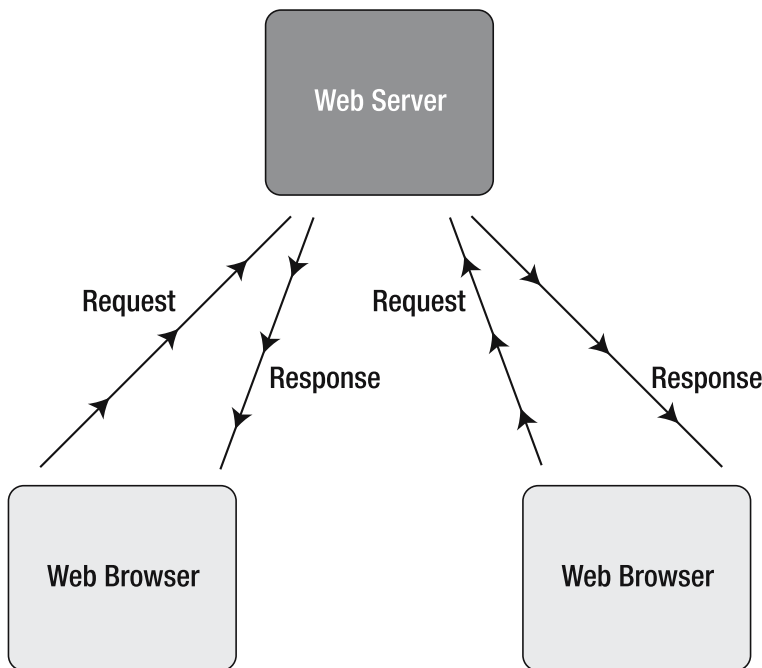


Figure 1-1. *The request/response method used in most web sites currently on the Internet.*

From CGI to Flash to DHTML

The development community has asked, and the corporations have answered. Developer tool after tool has been designed, each with its own set of pros and cons. Perhaps the first scripting language to truly allow web applications the freedom they were begging for was the server-side processing language CGI (Common Gateway Interface).

With the advent of CGI, developers could now perform complex actions such as—but certainly not limited to—dynamic image creation, database management, complex calculation, and dynamic web content creation. What we have come to expect from our web applications today started with CGI. Unfortunately, while CGI addressed many issues, the elusive problem of seamless interaction and response remained.

In an attempt to create actual living, breathing, moving web content, Macromedia (www.macromedia.com) released its highly functional, and rather breathtaking (for the time) Flash suite. Flash was, and still remains to this day, very aptly named. It allows a web developer to create visually impressive “movies” that can function as web sites, applications, and more. These web sites were considered significantly “flashier” than other web sites, due to their ability to have motion rendered all across the browser.

In the hands of a professional designer, Flash-enabled web sites can be quite visually impressive. Likewise, in the hands of a professional developer, they can be very powerful.

However, it's rare that an individual possesses both considerable design and development skills; therefore, Flash applications tend to be either visually impressive with very little functionality, or functionally amazing with an interface that leaves much to be desired. Also, this dilemma is combined with an additional compatibility issue: in order for Flash to function, a plug-in must be installed into your browser.

Another visually dynamic technology that has been around for many years but does not have a significant base of users is DHTML (an acronym for Dynamic HyperText Markup Language). DHTML—a term describing the marriage of JavaScript and HTML—basically combines HTML and CSS elements with JavaScript in an attempt to make things happen in your web browser dynamically. While DHTML in the hands of a skilled JavaScript professional can achieve some impressive results, the level of expertise required to do so tends to keep it out of the hands of most of the development community.

While scripts such as drop-down menus, rollovers, and tool tip pop-ups are fairly commonplace, it is only due to skilled individuals creating packages that the everyday developer can deploy. Very few individuals code these software packages from scratch, and up until recently, not many individuals considered JavaScript a very potent tool for the Internet.

Pros and Cons of Today's Web Application Environment

There are very obvious pros and cons to creating web applications on the Internet. While desktop applications continually struggle with cross-platform compatibility issues, often fraught with completely different rules for handling code, Internet applications are much simpler to port between browsers. Combine that with the fact that only a few large-scale browsers contain the vast majority of the user base, and you have a means of deployment that is much more stable across different users.

There is also the much-appreciated benefit to being able to create and maintain a single code base for an online application. If you were to create a desktop application and then deploy a patch for a bug fix, the user must either reinstall the entire software package or somehow gain access to the patch and install it. Furthermore, there can be difficulty in determining which installations are affected.

Web applications, on the other hand, can be located at one single server location and accessed by all. Any changes/improvements to the functionality can be delivered in one central location and take effect immediately. Far more control is left in the hands of the developers, and they can quite often continue to create and maintain a superior product.

Naturally, everything comes with a price. While delivering an application from a central server location is quite nice from a maintenance point of view, the problem arises that the client needs a means to access said point of entry. The Internet provides a wonderful way to do this, but the question of speed comes into play immediately.

While a client using Microsoft Word, for example, can simply click a button on their computer to fire it up and receive an instant response, applications built on the Internet require a connection to said application to use it. While high-speed Internet is gaining more and more ground every day, a vast majority of Internet users are still making use of the much slower 56 Kbps (and slower) modems. Therefore, even if the software can quickly process information on the server, it may take a considerable amount of time to deliver it to the end user.

Combine this issue with the need to refresh the page every time a server response is required, and you can have some very frustrating issues for the end user of an Internet application. A need is definitely in place for web applications that contain the benefits of deliverability with the speed of a desktop application. As mentioned, Flash provides such a means, to an extent, through its powerful ActionScript language, but you need to be a jack-of-all-trades to effectively use it. DHTML provides a means to do this through the use of JavaScript, but the code to do so is rather restrictive.

Even worse, you often have to deal with browsers that refuse to cooperate with a real set of standards (or rather, fail to follow the standards). Thankfully, though, there is a solution to these problems: Ajax. Dubbed Asynchronous JavaScript and XML by Jesse James Garrett, and made popular largely by such web applications as Google's Gmail, Ajax is a means to making server-side requests with seamless page-loading and little to no need for full page refreshes.

Enter Ajax

Ajax took the Internet world rather by surprise, not just in its ease of use and very cool functionality, but also in its ability to draw the attention of darn near every developer on the planet. Where two years ago Ajax was implemented rather dubiously, without any form of standard (and certainly there were very few sites that built their core around Ajax completely), Ajax is now seemingly as commonplace as the rollover.

Entire web applications are arising out of nowhere, completely based upon Ajax functionality. Not only are they rather ingenious uses of the technology, they are leading the web industry into a new age whereby the standard web browser can become so much more; it can even rival the desktop application now.

Take, for instance, Flickr (www.flickr.com) or Gmail (www.gmail.com) (see Figure 1-2). On their surface, both offer services that are really nothing new. (After all, how many online photo albums and web mail services are out there?) Why then have these two applications garnered so much press and publicity, particularly in the online community?

I believe the reason for the new popularity of Ajax-based applications is not that the functionality contained within is anything new or astounding; it is merely the fact that the way the information and functionality is presented to us is done in a very efficient and ergonomic manner (something that, up until now, has been largely absent within Internet applications).

Home | Tags | Groups | People | Invite Logged in as [lbabinz](#) | [Your Account](#) | [Help](#) | [Sign Out](#)

Photos: [Yours](#) • [Upload](#) • [Organize](#) • [Your Contacts](#) • [Explore](#) **flickr**^{BETA}

Hola lbabinz!

- ◆ [Choose your Flickr web address!](#)
- ◆ [Create yourself a buddy icon!](#)
- ◆ [Did you know you can email images to Flickr?](#)

Printing? Can it be true?
Well, it's true *if you're in the U.S.*... with more countries coming online soon! Get 10 free prints with your first order! [Click here to set yourself up for printing.](#)


Holy smokes! That's cheap!
Just **\$24.95** for a 1 year pro account.
[Find out how to upgrade your account.](#)
(There will always be a free version. [Learn More.](#))

Do you use RSS and Atom Feeds?
Find out what Flickr feeds you can subscribe to!


[Invite your friends and family](#)

» [Upload photos](#) (Or, look at our [uploading tools](#)...)

» [Your photos](#) ([Recent activity](#) / [Comments you've made](#))



» [Everyone's photos](#) ([Hide](#))



» [Your groups](#) ([See what's new](#))

Gmail^{by Google} Welcome to Gmail

A Google approach to email.

Gmail is an experiment in a new kind of webmail, built on the idea that you should never have to delete mail and you should always be able to find the message you want. The key features are:

- ◆ **Search, don't sort.**
Use Google search to **find the exact message** you want, no matter when it was sent or received.
- ◆ **Don't throw anything away.**
Over 2661.915847 megabytes (and counting) of free storage so you'll never need to delete another message.
- ◆ **Keep it all in context.**
Each message is grouped with all its replies and displayed as a conversation.
- ◆ **No pop-up ads. No untargeted banners.**
You see only [relevant text ads](#) and links to related web pages of interest.

Sign in to Gmail with your
Google Account

Username:

Password:

Remember me on this computer.

[Forgot your username or password?](#)

[Learn more about Gmail.](#)

Check out our [new features!](#)

A few words about [Gmail and privacy.](#)

Figure 1-2. Web sites such as Flickr and Gmail have created rich Ajax applications.

Ajax Defined

Ajax, as stated previously, stands for Asynchronous JavaScript and XML. Now, not everyone agrees that Ajax is the proper term for what it represents, but even those who are critical of the term cannot help but understand the implications it stands for and the widespread fame that the technology has received, partly as a result of its new moniker.

Basically, what Ajax does is make use of the JavaScript-based XMLHttpRequest object to fire requests to the web server asynchronously—or without having to refresh the page. (Figures 1-3 and 1-4 illustrate the difference between traditional and Ajax-based request/response models.) By making use of XMLHttpRequest, web applications can garner/send information to the server, have the server do any processing that needs to be handled, and then change aspects of the web page dynamically without the user having to move pages or change the location of their focus. You might think that by using the XMLHttpRequest object, all code response would have to return XML. While it certainly can return XML, it can also return just about anything you tell your scripting language to return.

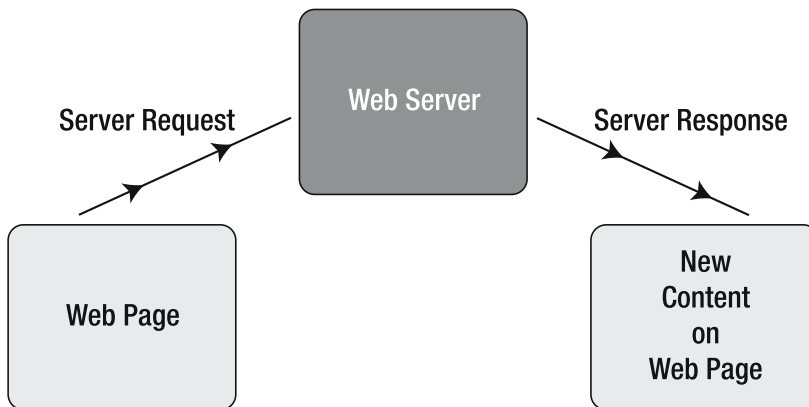


Figure 1-3. Traditional server request/response model used on most web-based applications today; each time a server request is made, the page must refresh to reveal new content

Consider, for instance, that you are using a mortgage calculator form to deduce the amount of money that is soon to be siphoned from your hard-earned bank account—not a trivial matter for your scripting language at all. The general way of handling such an application would be to fill out the form, press the submit button, and then wait for the response to come back. From there, you could redo the entire thing, testing with new financial figures.

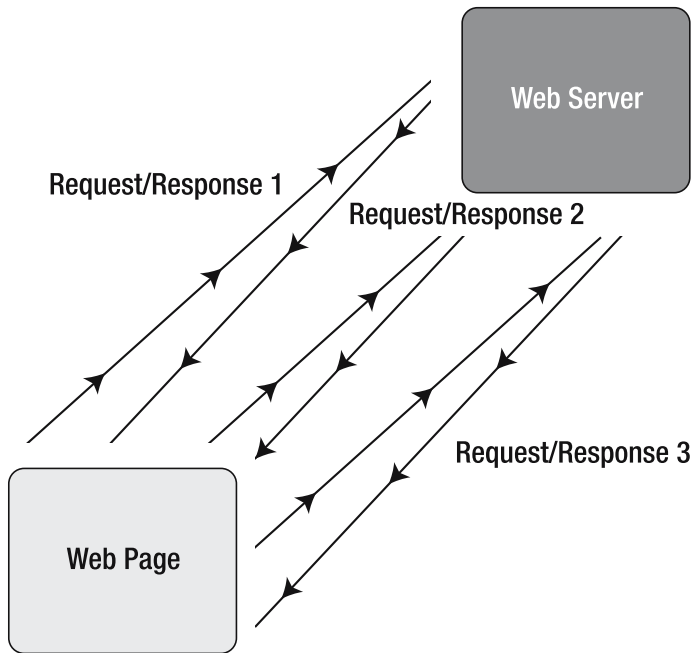


Figure 1-4. Internet request/response model using Ajax's asynchronous methodology; multiple server requests can be made from the page without need for a further page refresh

With a JavaScript-based Ajax solution, however, you could click the submit button, and while you remain fixed on the same page, the server could do the calculations and return the value of the mortgage right in front of your eyes. You could then change values in the formula and immediately see the differences.

Interestingly, new ergonomic changes can now occur as well. Perhaps you don't even want to use a submit button. You could use Ajax to make a call to the server every time you finished entering a field, and the number would adjust itself immediately. Ergonomic features such as this are just now becoming mainstream.

Is Ajax Technology New?

To call Ajax a new technology in front of savvy web developers will guarantee an earful of ranting. Ajax is not a new technology—in fact, Ajax is not even really a technology at all. Ajax is merely a term to describe the process of using the JavaScript-based XMLHttpRequest object to retrieve information from a web server in a dynamic manner (asynchronously).

The means to use the XMLHttpRequest has been prevalent as far back as 1998, and web browsers such as Internet Explorer 4 have possessed the capability to make use of Ajax even back then (albeit not without some configuration woes). Long before the browser you are likely using right now was developed, it was quite possible to make use of JavaScript to handle your server-side requests instantaneously from a client-side point of view.

However, if we are talking about the widespread use of Ajax as a concept (not a technology), then yes, it is quite a new revelation in the Internet community. Web developers of all kinds have finally started coming around to the fact that not all requests to the server have to be done in the same way. In some respects, Ajax has opened the minds of millions of web developers who were simply too caught up in convention to see beyond the borders of what is possible. Please do not consider me a pioneer in this respect either; I was one of them.

Why Ajax Is Catching Fire Now

So, if this technology has existed for so long, why is it only becoming so popular now? It is hard to say exactly why it caught fire in the first place, or who is to really be credited for igniting the fire under its widespread fame. Many developers will argue over Gmail and its widespread availability, or Jesse James Garrett for coining the term and subsequently giving people something to call the concept; but the true success of Ajax, I believe, lies more in the developers than in those who are using it.

Consider industries such as accounting. For years, accountants used paper spreadsheets and old-fashioned mathematics to organize highly complex financials. Then, with the advent of computers, things changed. A new way of deploying their services suddenly existed and the industry ceased to remain the way it once was. Sure, standards from the old way still hold true to this day, but so much more has been added, and new ways of doing business have been created.

Ajax has created something like this for Internet software and web site developers. Conventions that were always in place still remain, but now we have a new way to deploy functionality and present information. It is a new tool that we can use to do business with and refine our trade. New methodologies are now in place to deploy that which, up until very recently, seemed quite out of our grasp as developers. I, for one, am rather excited to be building applications using the Ajax concept, and can't wait to see what creative Internet machines are put into place.

Ajax Requirements

Since Ajax is based upon JavaScript technology, it goes without saying that JavaScript must be enabled in the user's browser in order for it to work. That being said, most people do allow their browsers to use JavaScript, and it is not really that much of a security issue to have it in place. It must be noted, however, that the user does have the ability to

effectively “disable” Ajax, so it is important to make sure, when programming an Ajax application, that other means are available to handle maneuvering through the web site; or alternatively, that the user of the web site is kept properly informed of what is necessary to operate the application.

Ajax is a fairly widely supported concept across browsers, and can be invoked on Firefox (all available versions), Internet Explorer (4.0 and higher), Apple Safari (1.2 and higher), Konqueror, Netscape (7.1 and higher), and Opera (7.6 and higher). Therefore, most browsers across the widely used gamut have a means for handling Ajax and its respective technologies. For a more complete listing on handling cross-browser Ajax, have a look at Chapter 11.

At this point, the only real requirement for making use of Ajax in an efficient and productive manner is the creativity of going against what the standard has been telling us for years, and creating something truly revolutionary and functional.

Summary

You should now have a much better understanding of where this upstart new technology has come from and where it intends to go in the future. Those web developers out there who are reading this and have not experimented yet with Ajax should be salivating to see what can be done. The first time I was introduced to the concept of running server requests without having to refresh the page, I merely stood there in awe for a few minutes running through all of the amazing ideas I could now implement. I stood dumbfounded in the face of all of the conventions this technology broke down.

Ready for more yet? Let's move on to the next chapter and start getting Ajax and PHP to work for you.



Ajax Basics

An interesting misconception regarding Ajax is that, given all the cool features it has to offer, the JavaScript code must be extremely difficult to implement and maintain. The truth is, however, that beginning your experimentation with the technology could not be simpler. The structure of an Ajax-based server request is quite easy to understand and invoke. You must simply create an object of the XMLHttpRequest type, validate that it has been created successfully, point where it will go and where the result will be displayed, and then send it. That's really all there is to it.

If that's all there is to it, then why is it causing such a fuss all of a sudden? It's because Ajax is less about the code required to make it happen and more about what's possible from a functionality, ergonomics, and interface perspective. The fact that Ajax is rather simple to implement from a development point of view is merely icing on a very fine cake. It allows developers to stop worrying about making the code work, and instead concentrate on imagining what might be possible when putting the concept to work.

While Ajax can be used for very simple purposes such as loading HTML pages or performing mundane tasks such as form validation, its power becomes apparent when used in conjunction with a powerful server-side scripting language. As might be implied by this book's title, the scripting language I'll be discussing is PHP. When mixing a client-side interactive concept such as Ajax with a server-side powerhouse such as PHP, amazing applications can be born. The sky is the limit when these two come together, and throughout this book I'll show you how they can be mixed for incredibly powerful results.

In order to begin making use of Ajax and PHP to create web applications, you must first gain a firm understanding of the basics. It should be noted that Ajax is a JavaScript tool, and so learning the basics of JavaScript will be quite important when attempting to understand Ajax-type applications. Let's begin with the basics.

HTTP Request and Response Fundamentals

In order to understand exactly how Ajax concepts are put together, it is important to know how a web site processes a request and receives a response from a web server. The current standard that browsers use to acquire information from a web server is the HTTP

(HyperText Transfer Protocol) method (currently at version HTTP/1.1). This is the means a web browser uses to send out a request from a web site and then receive a response from the web server that is currently in charge of returning the response.

HTTP requests work somewhat like e-mail. That is to say that when a request is sent, certain headers are passed along that allow the web server to know exactly what it is to be serving and how to handle the request. While most headers are optional, there is one header that is absolutely required (provided you want more than just the default page on the server): the `host` header. This header is crucial in that it lets the server know what to serve up.

Once a request has been received, the server then decides what response to return. There are many different response codes. Table 2-1 has a listing of some of the most common ones.

Table 2-1. *Common HTTP Response Codes*

Code	Description
200 OK	This response code is returned if the document or file in question is found and served correctly.
304 Not Modified	This response code is returned if a browser has indicated that it has a local, cached copy, and the server's copy has not changed from this cached copy.
401 Unauthorized	This response code is generated if the request in question requires authorization to access the requested document.
403 Forbidden	This response code is returned if the requested document does not have proper permissions to be accessed by the requestor.
404 Not Found	This response code is sent back if the file that is attempting to be accessed could not be found (e.g., if it doesn't exist).
500 Internal Server Error	This code will be returned if the server that is being contacted has a problem.
503 Service Unavailable	This response code is generated if the server is too overwhelmed to handle the request.

It should be noted that there are various forms of request methods available. A few of them, like `GET` and `POST`, will probably sound quite familiar. Table 2-2 lists the available request methods (although generally only the `GET` and `POST` methods are used).

Table 2-2. *HTTP Request Methods*

Method	Description
GET	The most common means of sending a request; simply requests a specific resource from the server
HEAD	Similar to a GET request, except that the response will come back without the response body; useful for retrieving headers
POST	Allows a request to send along user-submitted data (ideal for web-based forms)
PUT	Transfers a version of the file request in question
DELETE	Sends a request to remove the specified document
TRACE	Sends back a copy of the request in order to monitor its progress
OPTIONS	Returns a full list of available methods; useful for checking on what methods a server supports
CONNECT	A proxy-based request used for SSL tunneling

Now that you have a basic understanding of how a request is sent from a browser to a server and then has a response sent back, it will be simpler to understand how the `XMLHttpRequest` object works. It is actually quite similar, but operates in the background without the prerequisite page refresh.

The XMLHttpRequest Object

Ajax is really just a concept used to describe the interaction of the client-side `XMLHttpRequest` object with server-based scripts. In order to make a request to the server through Ajax, an object must be created that can be used for different forms of functionality. It should be noted that the `XMLHttpRequest` object is both instantiated and handled a tad differently across the browser gamut. Of particular note is that Microsoft Internet Explorer creates the object as an ActiveX control, whereas browsers such as Firefox and Safari use a basic JavaScript object. This is rather crucial in running cross-browser code as it is imperative to be able to run Ajax in any type of browser configuration.

XMLHttpRequest Methods

Once an instance of the `XMLHttpRequest` object has been created, there are a number of methods available to the user. These methods are expanded upon in further detail in Table 2-3. Depending on how you want to use the object, different methods may become more important than others.

Table 2-3. *XMLHttpRequest Object Methods*

Method	Description
<code>abort()</code>	Cancels the current request
<code>getAllResponseHeaders()</code>	Returns all HTTP headers as a String type variable
<code>getResponseHeader()</code>	Returns the value of the HTTP header specified in the method
<code>open()</code>	Specifies the different attributes necessary to make a connection to the server; allows you to make selections such as GET or POST (more on that later), whether to connect asynchronously, and which URL to connect to
<code>setRequestHeader()</code>	Adds a label/value pair to the header when sent
<code>send()</code>	Sends the current request

While the methods shown in Table 2-3 may seem somewhat daunting, they are not all that complicated. That being said, let's take a closer look at them.

abort()

The abort method is really quite simple—it stops the request in its tracks. This function can be handy if you are concerned about the length of the connection. If you only want a request to fire for a certain length of time, you can call the abort method to stop the request prematurely.

getAllResponseHeaders()

You can use this method to obtain the full information on all HTTP headers that are being passed. An example set of headers might look like this:

```
Date: Sun, 13 Nov 2005 22:53:06 GMT
Server: Apache/2.0.53 (win32) PHP/5.0.3
X-Powered-By: PHP/5.0.3
Content-Length: 527
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: text/html
```

getResponseHeader("headername")

You can use this method to obtain the content of a particular piece of the header. This method can be useful to retrieve one part of the generally large string obtained from a set of headers. For example, to retrieve the size of the document requested, you could simply call `getResponseHeader ("Content-Length")`.

open ("method","URL","async","username","pswd")

Now, here is where we start to get into the meat and potatoes of the `XMLHttpRequest` object. This is the method you use to open a connection to a particular file on the server. It is where you pass in the method to open a file (GET or POST), as well as define how the file is to be opened. Keep in mind that not all of the arguments in this function are required and can be customized depending on the situation.

setRequestHeader("label","value")

With this method, you can give a header a label of sorts by passing in a string representing both the label and the value of said label. An important note is that this method may only be invoked after the `open()` method has been used, and must be used before the `send` function is called.

send("content")

This is the method that actually sends the request to the server. If the request was sent asynchronously, the response will come back immediately; if not, it will come back after the response is received. You can optionally specify an input string as an argument, which is helpful for processing forms, as it allows you to pass the values of form elements.

XMLHttpRequest Properties

Of course, any object has a complete set of properties that can be used and manipulated in order for it work to its fullest. A complete list of the `XMLHttpRequest` object properties is presented in Table 2-4. It is important to take note of these properties—you will be making use of them as you move into the more advanced functionality of the object.