Phil Ballard
Michael Moncur

Sams **Teach Yourself**

Ajax, JavaScript and PHP

All in One

SAMS

Sams Teach Yourself Ajax, JavaScript, and PHP All in One

ISBN-13: 978-0-672-32965-4
ISBN-10: 0-672-32965-4

Library of Congress Cataloging-in-Publication Data

Ballard, Phil.

Sams teach yourself Ajax, JavaScript, and PHP all in one / Phil

Ballard, Michael Moncur.

p. cm.

Includes index.

ISBN 978-0-672-32965-4 (pbk. : CD-ROM)

1. Ajax (Web site development technology) 2. JavaScript (Computer program language) 3. PHP (Computer program language) 4. Web site development. I. Moncur, Michael G. II. Title. III. Title: Teach yourself Ajax, JavaScript, and PHP all in one.

TK5105.8885.A52B38 2008

006.7'6–dc22

2008022476

Printed in the United States of America

First Printing June 2008

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Editor-in-Chief
Mark Taub

Acquisitions Editor
Mark Taber

Managing Editor
Patrick Kanouse

Project Editor
Mandie Frank

Indexer
Ken Johnson

Proofreader
Paula Lowell

Publishing Coordinator
Vanessa Evans

Multimedia Developer
Dan Scherf

Designer
Gary Adair

Composition
TnT Design, Inc.

Introduction

Over the last decade or so, the World Wide Web has grown in scope from being a relatively simple information repository to becoming the first stop for many people when seeking entertainment, education, news, or business resources.

Websites themselves need no longer be limited to a number of static pages containing text and perhaps simple images; the tools now available allow the development of highly interactive and engaging pages involving animations, visual effects, context-sensitive content, embedded productivity tools, and much more.

The list of technologies available for producing such pages is broad. However, those based on Open Source licenses have become, and remain, highly popular due to their typically low (often zero) entry cost, and to the huge resource of user-contributed scripts, tutorials, tools, and other resources for these tools and applications available via the Internet and elsewhere.

In this book, we give a detailed account of how to program fluid, interactive websites using server- and client-side coding techniques and tools, as well as how to combine these to produce a slick, desktop-application-like user experience using Ajax.

The programming languages used in this book include the ubiquitous JavaScript (for client-side programming) and the immensely popular open-source PHP language (for server-side scripting, and available with the majority of web-hosting packages). The nuts and bolts of Ajax programming are described in detail, as well as the use of several advanced open-source frameworks that contain ready-written code for quickly building state-of-the-art interactive sites.

The CD that accompanies this book provides all the tools required on your journey through learning to program in PHP, JavaScript, and Ajax.

On the CD

What Is Ajax?

Ajax stands for *Asynchronous JavaScript And XML*. Although strictly speaking Ajax is not itself a technology, it mixes well-known programming techniques in an uncommon way to enable web developers to build Internet applications with much more appealing user interfaces than those to which we have become accustomed.

When using popular desktop applications, we expect the results of our work to be made available immediately, without fuss, and without our having to wait for the whole screen to be redrawn by the program. While using a spreadsheet such as Excel, for instance, we expect the changes we make in one cell to propagate immediately through the neighboring cells while we continue to type, scroll the page, or use the mouse.

Unfortunately, this sort of interaction has seldom been available to users of web-based applications. Much more common is the experience of entering data into form fields, clicking on a button or a hyperlink and then sitting back while the page slowly reloads to exhibit the results of the request. In addition, we often find that the majority of the reloaded page consists of elements that are identical to those of the previous page and that have therefore been reloaded unnecessarily; background images, logos, and menus are frequent offenders.

Ajax promises us a solution to this problem. By working as an extra layer between the user's browser and the web server, Ajax handles server communications in the background, submitting server requests and processing the returned data. The results may then be integrated seamlessly into the page being viewed, without that page needing to be refreshed or a new one being loaded.

In Ajax applications, such server requests are not necessarily synchronized with user actions such as clicking on buttons or links. A well-written Ajax application may already have asked of the server, and received, the data required by the user—perhaps before the user even knew she wanted it. This is the meaning of the *asynchronous* part of the Ajax acronym.

The parts of an Ajax application that happen "under the hood" of the user's browser, such as sending server queries and dealing with the returned data, are written in *JavaScript*, and *XML* is an increasingly popular means of coding and transferring formatted information used by Ajax to efficiently transfer data between server and client.

We'll look at all these techniques, and how they can be made to work together, as we work through the chapters.

Who This Book Is For

This volume is aimed primarily at web developers seeking to build better interfaces for the users of their web applications and programmers from desktop environments looking to transfer their applications to the Internet.

It also proves useful to web designers eager to learn how the latest techniques can offer new outlets for their creativity. Although the nature of PHP, JavaScript, and Ajax applications means that they require some programming, all the required technologies are explained from first principles within the book, so even those with little or no programming experience should be able to follow the lessons without a great deal of difficulty.

How To Use This Book

All the technologies—including a refresher of WWW basics—are explained from first principles, so that even non-programmers or those unfamiliar with these languages should be able to follow the development of the concepts with little problem.

The book is divided into parts, each dedicated to a particular technology or discussion topic. Within each part, the chapters each specialize in a given aspect or subtopic. It should therefore be easy to follow the instructional flow of the book by a quick look through the table of contents.

However, if you are already a competent programmer in one or more of the technologies used—in PHP for instance, or in JavaScript—then feel free to speed-read or skip the sections that you don't need.

To try out many of the examples you'll need access to a web server that supports PHP, and a means to upload files into your web space (probably FTP). Most web hosts include PHP in their hosting packages, or can do so on request at minimal or no cost.

Alternatively, the CD that accompanies this book contains everything required to set up a web serving environment on your own computer. This package is called XAMPP, and it contains everything you need to develop fully functional, interactive websites like those described in this book, ready to be deployed to a web-based server at a later date if you so choose. Look out for the boxes marked "On the CD" as you work through the book.

Conventions Used In This Book

This book contains special elements as described by the following:

These boxes highlight information that can make your programming more efficient and effective.

Did you Know?

By the Way

These boxes provide additional information related to material you just read.

Watch Out!

These boxes focus your attention on problems or side effects that can occur in specific situations.

▼

Try It Yourself

The Try It Yourself section offers suggestions for creating your own scripts, experimenting further, or applying the techniques learned throughout the chapter. This will help you create practical applications based on what you've learned.

▲

On the CD

Sections like this remind you about relevant information or tools available on the CD that accompanies the book.

A special `monospace` font is used on programming-related terms and language.

Setting Up Your Workspace

While you can write the code in this book using just a simple text editor, to run the examples you'll need a computer (with Windows, Mac, or Linux operating system) running a modern browser such as Internet Explorer or Firefox.

Did you know?

You can download Microsoft Windows Explorer from http://www.microsoft.com/ and the latest version of Firefox from http://www.mozilla.com/.

You will also need to load files on to a web server—if you already have a web host that supports PHP, you can use your web space there. Alternatively, the accompanying CD has everything you need to set up your own web server for private use, either on your own PC or another on your network.

What's on the CD

The accompanying CD contains everything you could need to get the best from this book. Included on the CD you'll find

- **XAMPP**, a complete open source compilation you can use to easily install the Apache web server, PHP language, and MySQL database manager on your computer. Versions are provided for Linux, Mac, and Windows environments.
- **jEdit**, a Java-based programmer's editor that's perfect for creating or modifying code. The CD includes files for Java, Mac, or Windows.
- A selection of **open source frameworks** for developing sophisticated web applications. Programming examples based on some of these frameworks are presented towards the end of the book.

CHAPTER 3

Anatomy of an Ajax Application

What You'll Learn in This Chapter:

- The Need for Ajax
- Introducing Ajax
- The Constituent Parts of Ajax
- Putting It All Together

In this chapter you will learn about the individual building blocks of Ajax and how they fit together to form the architecture of an Ajax application. Subsequent chapters will examine these components in more detail, finally assembling them into a working Ajax application.

The Need for Ajax

In the following parts of the book, we shall discuss each of the core components in detail.

Before discussing the individual components, though, let's look in more detail at what we want from our Ajax application.

Traditional Versus Ajax Client-Server Interactions

Chapter 1 discussed the traditional page-based model of a website user interface. When you interact with such a website, individual pages containing text, images, data entry forms, and so forth are presented one at a time. Each page must be dealt with individually before navigating to the next.

For instance, you may complete the data entry fields of a form, editing and re-editing your entries as much as you want, knowing that the data will not be sent to the server until the form is finally submitted.

Figure 3.1 illustrates this interaction.

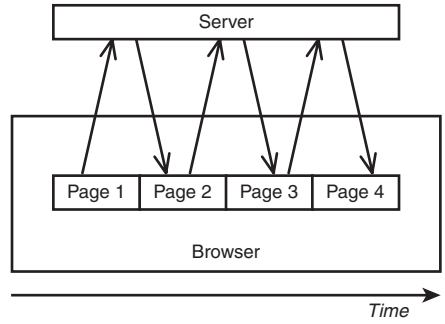

FIGURE 3.1 Traditional client–server interactions.

After you submit a form or follow a navigation link, you then must wait while the browser screen refreshes to display the new or revised page that has been delivered by the server.

As your experience as an Internet user grows, using this interface becomes almost second nature. You learn certain rules of thumb that help to keep you out of trouble, such as "don't click the Submit button a second time," and "don't click the Back button after submitting a form."

Unfortunately, interfaces built using this model have a few drawbacks. First, there is a significant delay while each new or revised page is loaded. This interrupts what we, as users, perceive as the "flow" of the application.

Furthermore, a *whole* page must be loaded on each occasion, even when most of its content is identical to that of the previous page. Items common to many pages on a website, such as header, footer, and navigation sections, can amount to a significant proportion of the data contained in the page.

Figure 3.2 illustrates a website displaying pages before and after the submission of a form, showing how much identical content has been reloaded and how relatively little of the display has actually changed.

This unnecessary download of data wastes bandwidth and further exacerbates the delay in loading each new page.

By the Way

Bandwidth refers to the capacity of a communications channel to carry information. On the Internet, bandwidth is usually measured in bps (bits per second) or in higher multiples such as Mbps (million bits per second).

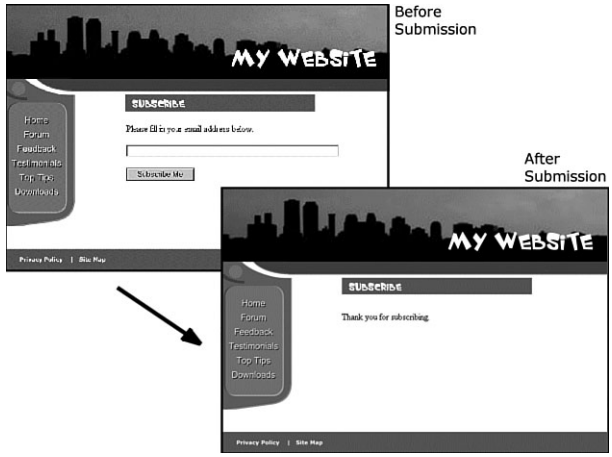

FIGURE 3.2
Many page items are reloaded unnecessarily.

The Rich User Experience

The combined effect of the issues just described is to offer a much inferior user experience compared to that provided by the vast majority of desktop applications.

On the desktop, you expect the display contents of a program to remain visible and the interface elements to respond to commands while the computing processes occur quietly in the background. As I write this chapter using a word processor, for example, I can save the document to disk, scroll or page up and down, and alter font faces and sizes without having to wait on each occasion for the entire display to be refreshed.

Ajax allows you to add to your web application interfaces some of this functionality more commonly seen in desktop applications and often referred to as a *rich user experience*.

Introducing Ajax

To improve the user's experience, you need to add some extra capabilities to the traditional page-based interface design. You want your user's page to be interactive, responding to the user's actions with revised content, and be updated without any interruptions for page loads or screen refreshes.

To achieve this, Ajax builds an extra layer of processing between the web page and the server.

This layer, often referred to as an *Ajax Engine* or *Ajax Framework*, intercepts requests from the user and in the background handles server communications quietly, unobtrusively, and *asynchronously*. By this we mean that server requests and responses no longer need to coincide with particular user actions but may happen at any time convenient to the user and to the correct operation of the application. The browser does not freeze and await the completion by the server of the last request but instead lets the user carry on scrolling, clicking, and typing in the current page.

The updating of page elements to reflect the revised information received from the server is also looked after by Ajax, happening dynamically while the page continues to be used.

Figure 3.3 represents how these interactions take place.

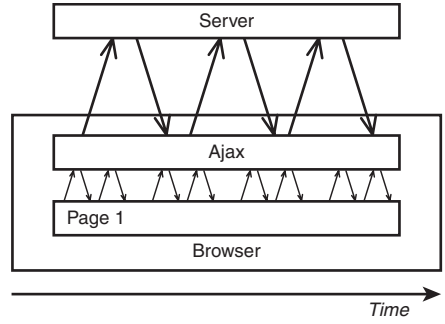

FIGURE 3.3
Ajax client–server interaction.

A Real Ajax Application—Google Suggest

To see an example of an Ajax application in action, let's have a look at *Google Suggest*. This application extends the familiar Google search engine interface to offer the user suggestions for suitable search terms, based on what he has so far typed.

With each key pressed by the user, the application's Ajax layer queries Google's server for suitably similar search phrases and presents the returned data in a drop-down box. Along with each suggested phrase is listed the number of results that would be expected for a search conducted using that phrase. At any point the user has the option to select one of these suggestions instead of continuing to type and have Google process the selected search.

Because the server is queried with every keypress, this drop-down list updates dynamically as the user types—with no waiting for page refreshes or similar interruptions.

Figure 3.4 shows the program in action. You can try it for yourself by following the links from Google's home page at http://www.google.com/webhp?complete=1&hl=en.

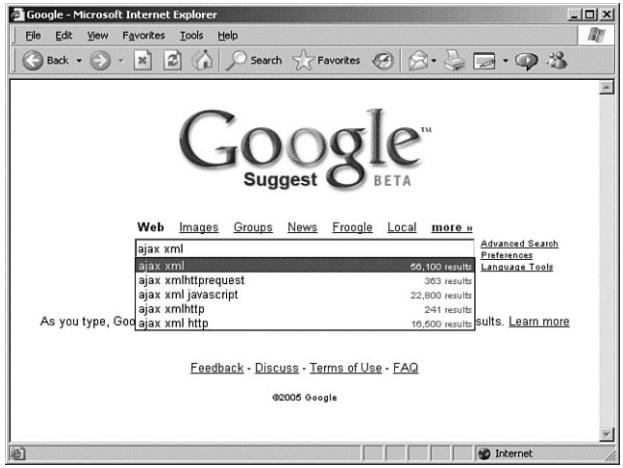

FIGURE 3.4 An example of an Ajax application—Google Suggest.

Next let's identify the individual components of such an Ajax application and see how they work together.

> Google has presented other Ajax-enabled applications that you can try, including the *gmail* web mail service and the *Google Maps* street mapping program. See the Google website at http://www.google.com/ for details.

By the Way

The Constituent Parts of Ajax

Now let's examine the components of an Ajax application one at a time.

The `XMLHTTPRequest` Object

When you click on a hyperlink or submit an HTML form, you send an HTTP request to the server, which responds by serving to you a new or revised page. For your web application to work asynchronously, however, you must have a means to send HTTP requests to the server *without* an associated request to display a new page.

You can do so by means of the `XMLHTTPRequest` *object*. This JavaScript object is capable of making a connection to the server and issuing an HTTP request without the necessity of an associated page load.

In following chapters you will learn what objects are, see how an instance of this object can be created, and see how its properties and methods can be used by JavaScript routines included in the web page to establish asynchronous communications with the server.

Did you Know?

As a security measure, the `XMLHTTPRequest` object can generally only make calls to URLs within the same domain as the calling page and cannot directly call a remote server.

Chapter 5, "Working with the Document Object Model" will introduce the concept of objects in general, and this subject will be expanded in Chapter 7 "Using Functions and Objects."

Chapter 10, "The Heart of Ajax"—the `XMLHTPPRequest` Object, discusses how to create an instance of the `XMLHTTPRequest` object and reviews the object's properties and methods.

Talking with the Server

In the traditional style of web page, when you issue a server request via a hyperlink or a form submission, the server accepts that request, carries out any required server-side processing, and subsequently serves to you a new page with content appropriate to the action you have undertaken.

While this processing takes place, the user interface is effectively frozen. You are made quite aware of this, when the server has completed its task, by the appearance in the browser of the new or revised page.

With asynchronous server requests, however, such communications occur in the background, and the completion of such a request does not necessarily coincide with a screen refresh or a new page being loaded. You must therefore make other arrangements to find out what progress the server has made in dealing with the request.

The `XMLHTTPRequest` object possesses a convenient property to report on the progress of the server request. You can examine this property using JavaScript routines to determine the point at which the server has completed its task and the results are available for use.

Your Ajax armory must therefore include a routine to monitor the status of a request and to act accordingly. We'll look at this in more detail in Chapter 11, "Talking with the Server."

What Happens at the Server?

So far as the server-side script is concerned, the communication from the `XMLHTTPRequest` object is just another HTTP request. Ajax applications care little about what languages or operating environments exist at the server; provided that the client-side Ajax layer receives a timely and correctly formatted HTTP response from the server, everything will work just fine.

It is possible to build simple Ajax applications with no server-side scripting at all, simply by having the `XMLHTTPRequest` object call a static server resource such as an XML or text file.

Ajax applications may make calls to various other server-side resources such as web services. Later in the book we'll look at some examples of calling web services using protocols such as SOAP and REST.

By the Way

In this book we'll be using the popular PHP scripting language for our server-side routines, but if you are more comfortable with ASP, JSP, or some other server-side language, go right ahead and use it in your Ajax applications.

Dealing with the Server Response

Once notified that an asynchronous request has been successfully completed, you may then utilize the information returned by the server.

Ajax allows for this information to be returned in a number of formats, including ASCII text and XML data.

Depending on the nature of the application, you may then translate, display, or otherwise process this information within the current page.

We'll look into these issues in Chapter 12, "Using the Returned Data."

Other Housekeeping Tasks

An Ajax application will be required to carry out a number of other duties, too. Examples include detecting error conditions and handling them appropriately, and keeping the user informed about the status of submitted Ajax requests.

You will see various examples in later chapters.

Putting It All Together

Suppose that you want to design a new Ajax application, or update a legacy web application to include Ajax techniques. How do you go about it?

First you need to decide what page events and user actions will be responsible for causing the sending of an asynchronous HTTP request. You may decide, for example, that the action of moving the mouse cursor over an image will result in a request being sent to the server to retrieve further information about the subject of the picture, or that the clicking of a button will generate a server request for information with which to populate the fields on a form.

JavaScript can be used to execute instructions on occurrences such as these, by employing event handlers. The details of how will be covered in detail in the following chapters. In your Ajax applications, such methods will be responsible for initiating asynchronous HTTP requests via `XMLHTTPRequest`.

Having made the request, you need to write routines to monitor the progress of that request until you hear from the server that the request has been successfully completed.

Finally, after receiving notification that the server has completed its task, you need a routine to retrieve the information returned from the server and apply it in the application. You may, for example, want to use the newly returned data to change the contents of the page's body text, populate the fields of a form, or pop open an information window.

Figure 3.5 shows the flow diagram of all this.

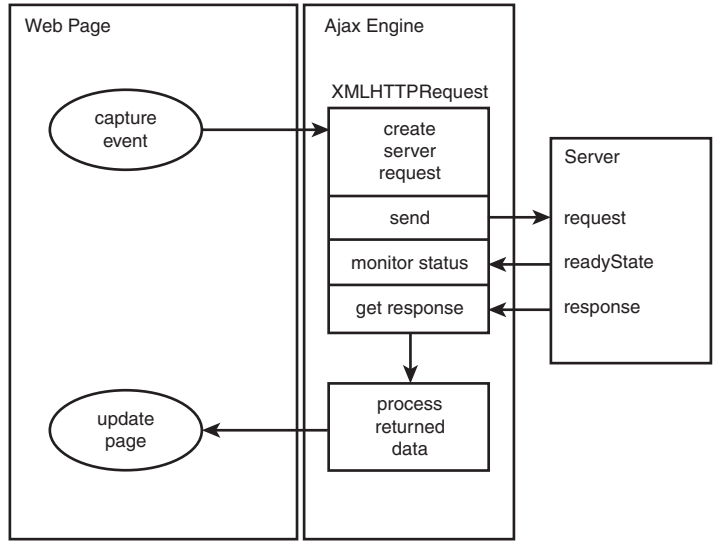

FIGURE 3.5
How the components of an Ajax application work together.

In Chapter 13, "Our First Ajax Application," you'll use what you have learned to construct a complete Ajax application.

Ajax Frameworks

While it is essential for a complete understanding of Ajax to understand what role each of the individual components plays, it is thankfully not necessary to rewrite all of your code for each new application. Your Ajax code can be stored as a reusable library of common Ajax routines, ready to be reused wherever they may be needed. There are also many commercial and open-source frameworks that you can use in your projects to do the "heavy lifting."

We shall look at both of these techniques later in the book, where we develop our own JavaScript library for Ajax, and also consider several of the more popular open-source libraries.

Summary

This chapter discussed the shortcomings of the traditional web interface, identifying specific problems we want to overcome. We also introduced the various building blocks of an Ajax application and discussed how they work together.

In the following chapters we shall look at these components in more detail, eventually using them to build a complete Ajax application.

That concludes Part I of the book, "Web Basics Refresher." In Part II we shall begin to explore client-side programming using JavaScript.

Index

SYMBOLS

A

B

C

D

E

F

G

H

J

K - L

M

N

O

P

Q - R

S

T

U

V

W

X

Y - Z