Sams Teach Yourself

JavaScript and Ajax

Video Learning Starter Kit

Quick Start Guide

Contents

Sams Teach Yourself JavaScript and Ajax: Video Learning Starter Kit

International Standard Book Number:

ISBN-13: 978-0-672-33037-7
ISBN-10: 0-672-33037-7

Printed in the United States of America

First Printing: April 2009

12 11 10 09 4 3 2 1

Reader Services

Visit our website and register this product at **informit.com/register** for convenient access to any updates, downloads or errata that may become available.

Before You Begin

This *Sams Teach Yourself Video Learning Starter Kit* is about the two web development technologies that provide the interactivity and visual effects at almost all current web sites and applications today.

JavaScript and Ajax are at work behind the scenes and on the homepages of everything from the simplest blog to huge sites like Facebook, Yahoo, and Google, with millions of subscribers and users worldwide.

While the videos in this course most likely won't show you how to develop the next Google Gmail or Yahoo homepage, they will give even non-programmers a solid understanding of how JavaScript and Ajax work and how you too can easily learn the basics of using JavaScript and Ajax, and begin to add interactivity and dynamic content to your own web site, large or small.

Before you get started watching the videos, however, we suggest that you first take a few minutes to review the material in this booklet. The information here will help you make sure you have all the equipment that's needed to watch the videos and do the exercises, and it will help you make best use of all the unique features of these courses as well as the software that's also on the DVD.

About JavaScript and Ajax

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

Web sites 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 standards 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 video course, we give a detailed account of how to program fluid, interactive web sites using server- and client-side coding techniques and tools, as well as combining 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.

About The Video Lessons

Sams Teach Yourself Video Learning courses are designed not only for beginners, but also for busy, time-crunched people who aren't able to devote the large blocks of time often required to take a full class on a subject, or to read some thick tome that covers everything in more detail than you'll ever need.

Each lesson in this course takes no more that 10 minutes to complete three basic steps:

1. Watch the video tutorial.
2. Try the hands-on examples in the lab section.
3. Take a short quiz to test yourself before moving on.

The short video tutorials clearly explain key concepts, terms, and techniques, and are rich in diagrams and clear, real-world examples. Plus, you can stop, start, pause or replay any part of the video you want, as often as you want, until you feel you really understand it.

After you've watched the video tutorial, you then you get to try it yourself and put what you've learned to work in a unique hands-on learning lab that helps you begin to develop real live programs that you can use on your own web site.

The software that you need to complete the exercises is included on the DVD. And as you continue to watch the lesson on your computer, you're guided step by step through the process of creating your own usable code.

Once you're done trying things out in the learning lab each lesson ends with a short, easy quiz so you can test what you've learned before you go on to the next lesson.

Who This Video Learning Kit is For

This course 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 JavaScript, Ajax, and PHP applications means that they require some programming, all of 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.

This course does *not* teach HTML. And although a thorough knowledge of HTML is not necessarily a prerequisite, the course does assumes a basic understanding of how web pages are constructed, where they’re stored and maintained, and how a user interacts with them through a web browser.

What’s on the DVD

The DVD for *Sams Teach Yourself JavaScript and Ajax Video Learning* has several components—all accessible by either a Windows or a Mac computer:

- **25 ten-minute video lessons**—each made up of a short video tutorial, a hands-on learning lab for trying things out, and a short self-assessment quiz at the end.
- **Software for the learning labs**—an easy-to-install package to set up a PHP- and MySQL-enabled Apache server on your computer, a script editing program, three JavaScript and Ajax libraries, and source code that you can cut-and-paste or customize in your own scripts.
- **The complete text of the lessons in PDF**—All the video lessons in this course are based on chapters from the print book *Sams Teach Yourself Ajax, JavaScript and PHP All in One*. So to help you out if you want to re-read a particular lesson or quickly look up a term or technique, we’ve included the full text of the book in easily searchable PDF format.

How the Course Is Organized

Sams Teach Yourself JavaScript and Ajax Video Learning is organized into five parts, with each part made up of three to six lessons, each designed to be completed in 10 minutes or less.

Part I: Web Basics Refresher

The lessons in Part I provide a refresher on the fundamental building blocks of Web development:

- **Lesson 1: Workings of the Web.** Introduces you to what the World Wide Web is and its key HTTP protocol.
- **Lesson 2: Writing Pages in HTML and CSS.** Provides you with a sound foundation in HTML and CSS for the Ajax applications in later lessons.
- **Lesson 3: Anatomy of an Ajax Application.** Examines the individual building blocks of Ajax and how they fit together.

Part II: Web Scripting with JavaScript

The lessons in Part II covers the basics of adding scripting to web pages using JavaScript:

- **Lesson 4: Creating Simple Scripts in JavaScript.** Shows you how to create a simple script, edit it, and test it using a web browser.
- **Lesson 5: Working with the DOM.** Introduces one of the most important tools you'll use with JavaScript: the Document Object Model (DOM).
- **Lesson 6: Variables, Strings, and Arrays.** Explains three tools for storing data in JavaScript: variables, which store numbers or text; strings, which are special variables for working with text; and arrays, which are multiple variables you can refer to by number.
- **Lesson 7: Using Functions and Objects.** Covers two more key JavaScript concepts you'll use later on with Ajax: functions, which enable you to group any number of statements into a block, and objects, which enable you to group data.
- **Lesson 8: Conditions and Loops – Controlling Flow.** Examines two ways to control program flow in JavaScript: conditions, which allow a choice of difference options depending on a value, and loops, which allow repetitive statements.

- **Lesson 9: Using Built-In Functions and Third-Party Libraries.** Covers the use of some key objects in JavaScript, including Math and Date, as well as third-party libraries

Part III: Introducing Ajax

The lessons in Part III introduce Ajax technologies for extending JavaScript's capabilities:

- **Lesson 10: The `XMLHTTPRequest` Object.** Examines the object at the heart of every Ajax application — the `XMLHTTPRequest` object.
- **Lesson 11: Talking with the Server.** Shows how to send requests to, and receive data from, the server.
- **Lesson 12: Using the Returned Data.** Examines how to process information returned from the server in response to an Ajax request.
- **Lesson 13: Our First Ajax Application.** Shows how to construct a complete and working Ajax application.

Part IV: Server-Side Scripting with PHP

The lessons in Part IV deal with using PHP to communicate with a web server and extend Ajax applications:

- **Lesson 14: Getting to Know PHP.** Tells you what PHP is all about and what it's able to do.
- **Lesson 15: Variables.** Examines how to assign values to variables in PHP and how to use them in some simple expressions.
- **Lesson 16: Flow Control.** Looks at how to control the flow of a PHP script using conditional statements and loops
- **Lesson 17: Functions.** Explains how frequently used sections of code can be turned into reusable functions.
- **Lesson 18: Using Classes.** Covers the basics of object-oriented PHP

Part V: Advanced Ajax Programming

Part IV looks at some advanced Ajax techniques for handling data and at building a library of Ajax scripts

- **Lesson 19: Returning Data as Text.** Shows how to use the `responseText` property to add functionality to Ajax applications.
- **Lesson 20: Asynchronous HTML and HTTP (AHAH).** Explains how to build Ajax-style applications without using XML.

- **Lesson 21: Returning Data as XML.** Shows how to use XML data returned from the server via the `responseXML` property of the `XMLHTTPRequest` object
- **Lesson 22: Implementing Web Services with REST and SOAP.** Discusses the basics of web services and how to implement them using the REST and SOAP protocols.
- **Lesson 23: Building an Ajax Library.** Explains how to build a small library of scripts that you can call from your applications.
- **Lesson 24: Avoiding Ajax Gotchas.** Discusses some of the common Ajax mistakes and how to avoid them.

1

Using this Video Learning Kit

The *Sams Teach Yourself JavaScript and Ajax Video Learning Kit* can be used in a variety of different ways, depending on what your learning needs and style might be.

Watching the video lessons alone would provide a good, quick overview of the JavaScript and Ajax and the process of developing simple Ajax applications. And the quizzes at the end of each lesson provide an easy way to quickly test yourself before you go on to the next lesson.

Both the videos and the quizzes require nothing more than a DVD drive on your computer and a web browser with the Flash plugin installed.

For the best, most complete experience, however, nothing beats learning by doing. The Learning Lab exercises in each lesson are designed to give the learner hands-on experience in a real, live (not simulated) web server environment.

All that’s required for the Learning Lab exercises is either a web hosting service that includes access to a web server and PHP, or taking the time before you begin the lessons to set up a web server learning environment on your Windows PC or Mac using the software included on the DVD.

NOTE: Make sure you've put the DVD in your computer's *DVD* drive. The DVD will not run on a computer's CD drive. Nor will it work in a DVD player designed for plugging into a TV for watching movies.

How to Watch the Videos

The requirements for watching the video lessons in this kit are fairly straightforward:

- Any kind of a Windows, Mac, or Linux PC with a DVD drive that's capable of reading data DVDs.
- A web browser, such as Microsoft Internet Explorer, Safari, or Mozilla Firefox.
- The Adobe Flash Player plugin, which allows you to view Flash videos in your browser. If you can watch videos on YouTube, you should be all set.

Once you're sure you've got everything, you're ready to get started:

1. Insert the DVD from this package into your DVD drive on your computer.
2. If the DVD main interface does not automatically start up, navigate to the DVD folder on your computer and double-click on the icon or file named **Start** (**start.exe** on Windows or **start.app** on Mac OS X).

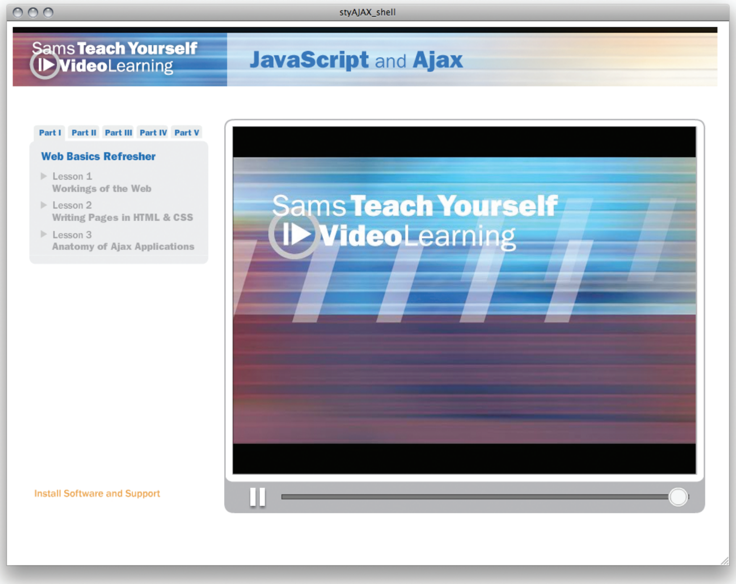

styAJAX_shell
Sams Teach Yourself VideoLearning
JavaScript and Ajax
Part I
Part II
Part III
Part IV
Part V
Web Basics Refresher
Lesson 1
Workings of the Web
Lesson 2
Writing Pages in HTML & CSS
Lesson 3
Anatomy of Ajax Applications
Install Software and Support
Sams Teach Yourself VideoLearning

FIGURE 1.1 The DVD's main interface.

3. To watch the videos, choose the lesson you want to start with from the list of lessons on the left side of the window (Figure 1.1), and then click on the play button in the player window on the right side.

How to Use the Learning Labs and Take the Quizzes

After each video in complete you can optionally work through the Learning Lab exercises and take a short (not too hard) quiz to help assess what you've learned in that lesson.

The Learning Lab exercises in this course require you to set up a web server learning environment either at a web hosting service or on your own computer (see the next section for detailed installation instructions).

Once your learning environment is set up, the instructions for each exercise walk you through the process one step at a time.

It's usually recommended that you take the time to type in the needed code for each exercise—it's generally thought that typing it in yourself helps you become more familiar with the details of how the code is structured and used. But if you're just a terrible typist or if you want to speed things up a little, the complete code for the exercise is on the second tab in the browser windows, ready for cutting and pasting into your code editor.

The quizzes after the exercises are relatively short and painless. After you make your choice, the program will tell you immediately if you're correct or wrong. Your answers are not stored or compiled, so you can take the quizzes as often as you like, and guessing won't count against you.

Setting up Your Learning Environment

As mentioned above, to complete the exercises in the lessons you'll need to have access to a web server environment running either on your own computer at home or on a web hosting service where you have access to the PHP scripting language and you can create and administer MySQL databases.

Many web hosting services offer PHP and MySQL access for customers who need them. If you're uncertain about this, inquire with your service whether they offer PHP and MySQL or not, and if so how to use them on your service.

However, since you're only going to be trying out simple exercises, and not presumably making your work available to the entire World Wide Web for now, a better solution for your learning environment might be to install the needed software on your computer using the DVD in this package. You don't even need to be connected to the Internet to do this.

The accompanying DVD contains a complete software toolkit with everything you'll need to complete the course's examples:

- **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 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.
- **Prototype**, **Scriptaculous**, **Rico**, and **XOAD**—popular JavaScript libraries for creating Ajax applications and effects

You can install this software by clicking on the "Install Software and Support" link on the opening screen of the DVD's main interface (Figure 1.1), or by navigating to the DVD in your computer's file browser and running the file named **Software** (**Software.exe** in Windows or **Software.app** in Mac OS X).

Once you've launched the software installer just follow the instructions on the screen to install either the complete packages or parts of it.

Web Basics Refresher

WHAT'S COVERED:

- Workings of the Web
- Writing Web Pages in HTML
- Anatomy of an Ajax Application

This chapter provides a quick, high-level refresher on the inner workings of the web, the mechanics of how web pages are constructed with HTML and CSS, and on the additional features that JavaScript and Ajax can bring to your web pages.

The information contained in this chapter is a prerequisite for getting the most out of the video lessons that make up the *Sams Teach Yourself JavaScript and Ajax Video Learning* course. If you have a solid background in HTML and web programming already, you can probably skip or just skim this material. If not, then read on.

Workings of the Web

We have a lot of ground to cover, so let's get to it. We'll begin by reviewing what the World Wide Web is and where it came from. Afterward we'll take a look at some of the major components that make it work, especially the HTTP protocol used to request and deliver web pages.

A Short History of the Internet

In the late 1950s, the U.S. government formed the Advanced Research Projects Agency (ARPA). This was largely a response to the Russian success in launching the *Sputnik* satellite and employed some of the country's top scientific intellects in research work with U.S. military applications.

During the 1960s, the agency created a decentralized computer network known as ARPAnet. This embryonic network initially linked four computers located at the University of California at Los Angeles, Stanford Research Institute, the University of California at Santa Barbara, and the University of Utah, with more nodes added in the early 1970s.

The network had initially been designed using the then-new technology of packet switching and was intended as a communication system that would remain functional even if some nodes should be destroyed by a nuclear attack.

Email was implemented in 1972, closely followed by the telnet protocol for logging on to remote computers and the File Transfer Protocol (FTP), enabling file transfer between computers.

This developing network was enhanced further in subsequent years with improvements to many facets of its protocols and tools. However, it was not until 1989 when Tim Berners-Lee and his colleagues at the European particle physics laboratory CERN (*Conseil Europeen pour le Recherche Nucleaire*) proposed the concept of linking documents with *hypertext* that the now familiar World Wide Web began to take shape. The year 1993 saw the introduction of Mosaic, the first graphical web browser and forerunner of the famous Netscape Navigator.

The use of hypertext pages and hyperlinks helped to define the page-based interface model that we still regard as the norm for web applications today.

The World Wide Web

The World Wide Web operates using a client/server networking principle. When you enter the URL (the web address) of a web page into your browser and click on "Go", you ask the browser to make an *HTTP request* of the particular computer having that address. On receiving this request, that computer returns ("serves") the required page to you in a form that your browser can interpret and display. Figure 2.1 illustrates this relationship. In the case of the Internet, of course, the server and client computers may be located anywhere in the world.

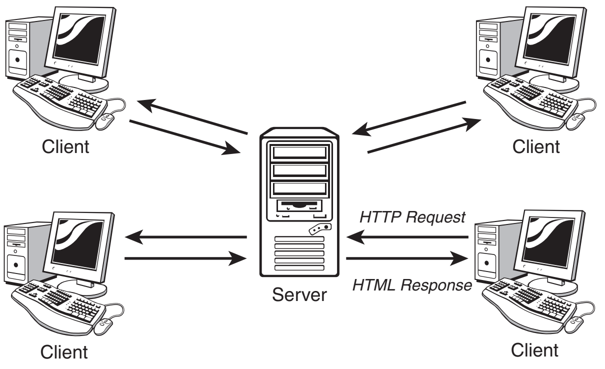


FIGURE 2.1 How web servers and clients (browsers) interact.

Later we'll discuss the nitty-gritty of HTTP requests in more detail. For now, suffice to say that your HTTP request contains several pieces of information needed so that your page may be correctly identified and served to you, including the following:

- The domain at which the page is stored (for example, mydomain.com)
- The name of the page (This is the name of a file in the web server's file system—for example, mypage.html)
- The names and values of any parameters that you want to send with your request

What Is a Web Page?

Anyone with some experience using the World Wide Web will be familiar with the term *web page*. The traditional user interface for websites involves the visitor navigating among a series of connected *pages* each containing text, images, and so forth, much like the pages of a magazine.

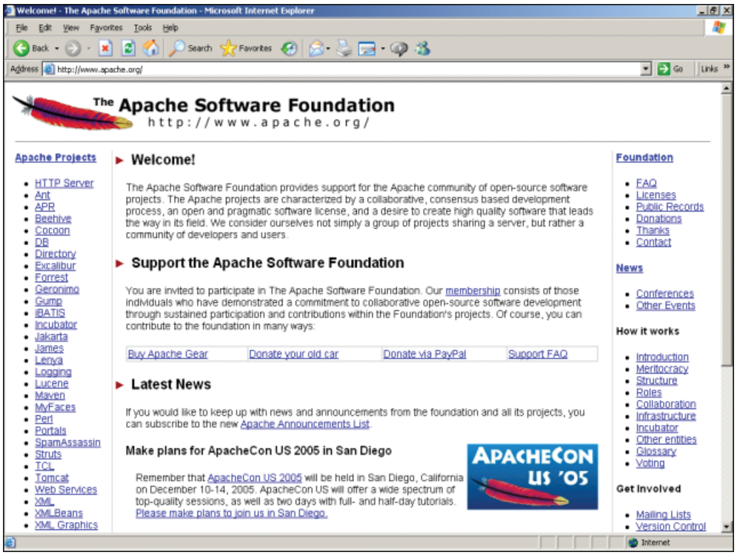
Generally speaking, each web page is actually a separate file on the server. The collection of individual pages constituting a website is managed by a program called a *web server*.

CAUTION: The term *web server* is often used in popular speech to refer to both the web server program—such as Apache—and the computer on which it runs.

Web Servers

A web server is a program that interprets HTTP requests and delivers the appropriate web page in a form that your browser can understand. Many examples are available, most running under either UNIX/Linux operating systems or under some version of Microsoft Windows.

Perhaps the best-known server application is the *Apache Web Server* from the Apache Software Foundation (http://www.apache.org), an open source project used to serve millions of websites around the world (see Figure 2.2).


Welcome! - The Apache Software Foundation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.apache.org/
Go Links
Internet

The Apache Software Foundation
http://www.apache.org/

Apache Projects

- HTTP Server
- Ant
- APR
- Beehive
- Cocoon
- DB
- Directory
- Excalibur
- Forrest
- Geronimo
- Gump
- iBATIS
- Incubator
- Jakarta
- James
- Lenya
- Logging
- Lucene
- Maven
- MyFaces
- Perl
- Portals
- SpamAssassin
- Struts
- TCL
- Tomcat
- Web Services
- XML
- XMLBeans
- XML Graphics

Welcome!

The Apache Software Foundation provides support for the Apache community of open-source software projects. The Apache projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license, and a desire to create high quality software that leads the way in its field. We consider ourselves not simply a group of projects sharing a server, but rather a community of developers and users.

Support the Apache Software Foundation

You are invited to participate in The Apache Software Foundation. Our membership consists of those individuals who have demonstrated a commitment to collaborative open-source software development through sustained participation and contributions within the Foundation's projects. Of course, you can contribute to the foundation in many ways:

Buy Apache Gear	Donate your old car	Donate via PayPal	Support FAQ

Latest News

If you would like to keep up with news and announcements from the foundation and all its projects, you can subscribe to the new Apache Announcements List.

Make plans for ApacheCon US 2005 in San Diego

Remember that ApacheCon US 2005 will be held in San Diego, California on December 10-14, 2005. ApacheCon US will offer a wide spectrum of top-quality sessions, as well as two days with full- and half-day tutorials. Please make plans to join us in San Diego.

APACHECON US '05

Foundation

- FAQ
- Licenses
- Public Records
- Donations
- Thanks
- Contact

News

- Conferences
- Other Events

How it works

- Introduction
- Meritocracy
- Structure
- Roles
- Collaboration
- Infrastructure
- Incubator
- Other entities
- Glossary
- Voting

Get Involved

- Mailing Lists
- Version Control

FIGURE 2.2 The Apache Software Foundation home page at http://www.apache.org/ displayed in Internet Explorer.

NOTE: Not all Windows-based web hosts use IIS. Various other web servers are available for Windows, including a version of the popular Apache Web Server.

Another example is Microsoft's IIS (Internet Information Services), often used on host computers running the Microsoft Windows operating system.

Server-Side Programming

Server-side programs, scripts, or languages, refer to programs that run on the server computer. Many languages and tools are available for server-side programming, including PHP, Java, and ASP (the latter being available only on servers running the Microsoft Windows operating system). Sophisticated server setups often also include databases of information that can be addressed by server-side scripts.

The purposes of such scripts are many and various. In general, however, they all are designed to preprocess a web page before it is returned to you. By this we mean that some or all of the page content will have been modified to suit the context of your request—perhaps to display train times to a particular destination and on a specific date, or to show only those products from a catalog that match your stated hobbies and interests.

NOTE: Server-side programming in this course is carried out using the popular PHP scripting language, which is flexible, is easy to use, and can be run on nearly all servers. Ajax, however, can function equally well with any server-side scripting language.

In this way server-side scripting allows web pages to be served with rich and varied content that would be beyond the scope of any design using only static pages—that is, pages with fixed content.

Web Browsers

A *web browser* is a program on a web surfer's computer that is used to interpret and display web pages. The first graphical web browser, Mosaic, eventually developed into the famous range of browsers produced by Netscape.

NOTE: By *graphical* web browser we mean one that can display not only the text elements of an HTML document but also images and colors. Typically, such browsers have a point-and-click interface using a mouse or similar pointing device.

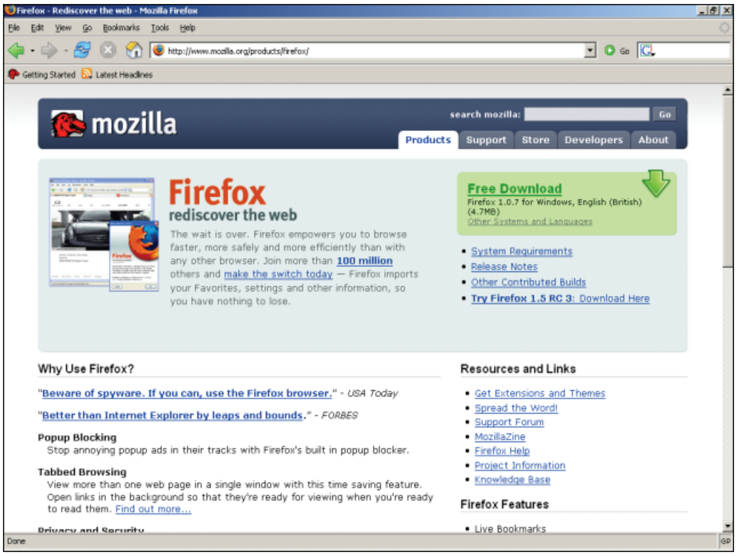
There also exist text-based web browsers, the best known of which is Lynx (http://lynx.browser.org/), which display HTML pages on character-based displays such as terminals, terminal emulators, and operating systems with command-line interfaces such as DOS.

The Netscape series of browsers, once the most successful available, were eventually joined by Microsoft's Internet Explorer offering, which subsequently went on to dominate the market.

Recent competitive efforts, though, have introduced a wide range of competing browser products including Opera, Safari, Konqueror, and especially Mozilla's Firefox, an open source web browser that has recently gained an enthusiastic following (see Figure 2.3).

Browsers are readily available for many computer operating systems, including the various versions of Microsoft Windows, UNIX/Linux, and

Macintosh, as well as for other computing devices ranging from mobile telephones to PDAs (Personal Digital Assistants) and pocket computers.


Firefox - Rediscover the web - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.mozilla.org/products/firefox/
Go
Getting Started
Latest Headlines
mozilla
search mozilla:
Go
Products
Support
Store
Developers
About
Done

Firefox
rediscover the web

The wait is over. Firefox empowers you to browse faster, more safely and more efficiently than with any other browser. Join more than 100 million others and make the switch today — Firefox imports your Favorites, settings and other information, so you have nothing to lose.

Free Download
Firefox 1.0.7 for Windows, English (British) (4.7MB)
Other Systems and Languages

- System Requirements
- Release Notes
- Other Contributed Builds
- Try Firefox 1.5 RC 3: Download Here

Why Use Firefox?

"Beware of spyware. If you can, use the Firefox browser." - *USA Today*

"Better than Internet Explorer by leaps and bounds." - *FORBES*

Popup Blocking
Stop annoying popup ads in their tracks with Firefox's built in popup blocker.

Tabbed Browsing
View more than one web page in a single window with this time saving feature. Open links in the background so that they're ready for viewing when you're ready to read them. Find out more...

Privacy and Security

Resources and Links

- Get Extensions and Themes
- Spread the Word!
- Support Forum
- MozillaZine
- Firefox Help
- Project Information
- Knowledge Base

Firefox Features

- Live Bookmarks

FIGURE 2.3 The Firefox browser from Mozilla.org browsing the Firefox Project home page.

Client-Side Programming

We have already discussed how server scripts can improve your web experience by offering pages that contain rich and varied content created at the server and inserted into the page before it is sent to you.

Client-side programming, on the other hand, happens not at the server but right inside the user's browser *after* the page has been received. Such scripts allow you to carry out many tasks relating to the data in the received page, including performing calculations, changing display colors and styles, checking the validity of user input, and much more.

Nearly all browsers support some version or other of a client-side scripting language called JavaScript, which is an integral part of Ajax and is the language we'll be using here for client-side programming.

DNS—The Domain Name Service

Every computer connected to the Internet has a unique numerical address (called an *IP address*) assigned to it. However, when you want to view a particular website in your browser, you don't generally want to type in a series of numbers—you want to use the domain name of the site in question. After all, it's much easier to remember www.somedomain.com than something like 198.105.232.4.

When you request a web page by its domain name, your Internet service provider submits that domain name to a DNS server, which tries to look up the database entry associated with the name and obtain the corresponding IP address. If it's successful, you are connected to the site; otherwise, you receive an error.

The many DNS servers around the Internet are connected together into a network that constantly updates itself as changes are made. When DNS information for a website changes, the revised address information is propagated throughout the DNS servers of the entire Internet, typically within about 24 hours.

Introducing HTTP

Various protocols are used for communication over the World Wide Web, perhaps the most important being *HTTP*, the protocol that is also fundamental to Ajax applications.

TIP: For a detailed account of HTTP, Sams Publishing offers the *HTTP Developer's Handbook* by Chris Shiflett.

When you request a web page by typing its address into your web browser, that request is sent using HTTP. The browser is an *HTTP client*, and the web page server is (unsurprisingly) an *HTTP server*.

In essence, HTTP defines a set of rules regarding how messages and other data should be formatted and exchanged between servers and browsers.

The HTTP Request and Response

The HTTP protocol can be likened to a conversation based on a series of questions and answers, which we refer to respectively as *HTTP requests* and *HTTP responses*.

The contents of HTTP requests and responses are easy to read and understand, being near to plain English in their syntax.

This section examines the structure of these requests and responses, along with a few examples of the sorts of data they may contain.

The HTTP Request

After opening a connection to the intended server, the HTTP client transmits a request in the following format:

- An opening line
- Optionally, a number of *header lines*
- A blank line
- Optionally, a message body

The opening line is generally split into three parts; the name of the *method*, the path to the required *server resource*, and the *HTTP version* being used. A typical opening line might read:

```
GET /sams/testpage.html HTTP/1.0
```

In this line we are telling the server that we are sending an HTTP request of type `GET` (explained more fully in the next section), we are sending this using HTTP version 1.0, and the server resource we require (including its local path) is

`/sams/testpage.html.`

NOTE: In this example the server resource we seek is on our own server, so we have quoted a relative path. It could of course be on another server elsewhere, in which case the server resource would include the full URL.

Header lines are used to send information about the request, or about the data being sent in the message body. One parameter and value pair is sent per line, the parameter and value being separated by a colon. Here's an example:

```
User-Agent: [name of program sending request]
```

For instance, Internet Explorer v5.5 offers something like the following:

```
User-agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
```

A further example of a common request header is the `Accept:` header, which states what sort(s) of information will be found acceptable as a response from the server:

```
Accept: text/plain, text/html
```

By issuing the header in the preceding example, the request is informing the server that the sending application can accept either plain text or HTML responses (that is, it is not equipped to deal with, say, an audio or video file).

The HTTP Response

In answer to such a request, the server typically issues an HTTP response, the first line of which is often referred to as the *status line*. In that line the server echoes the HTTP version and gives a response status code (which is a three-digit integer) and a short message known as a *reason phrase*. Here's an example HTTP response:

```
HTTP/1.0 200 OK
```

The response status code and reason phrase are essentially intended as machine-and human-readable versions of the same message, though the reason phrase may actually vary a little from server to server. Table 2.1 lists some examples of common status codes and reason phrases. The first digit of the status code usually gives some clue about the nature of the message:

- 1**—Information
- 2**—Success
- 3**—Redirected
- 4**—Client error
- 5**—Server error

NOTE: HTTP request methods include `POST`, `GET`, `PUT`, `DELETE`, and `HEAD`. By far the most interesting for the purposes here are the `GET` and `POST` requests. The `PUT`, `DELETE`, and `HEAD` requests are not covered here.

TABLE 2.1 Some Commonly Encountered HTTP Response Status Codes

STATUS CODE	EXPLANATION
`200 - OK`	The request succeeded.
`204 - No Content`	The document contains no data.
`301 - Moved Permanently`	The resource has permanently moved to a different URI.
`401 - Not Authorized`	The request needs user authentication.
`403 - Forbidden`	The server has refused to fulfill the request.
`404 - Not Found`	The requested resource does not exist on the server.
`408 - Request Timeout`	The client failed to send a request in the time allowed by the server.
`500 - Server Error`	Due to a malfunctioning script, server configuration error or similar.

TIP: A detailed list of status codes is maintained by the World Wide Web Consortium, W3C, and is available at http://www.w3.org/Protocols/rfc2616/ rfc2616-sec10.html.

The response may also contain header lines each containing a header and value pair similar to those of the HTTP request but generally containing information about the server and/or the resource being returned:

```
Server: Apache/1.3.22
Last-Modified: Fri, 24 Dec 1999 13:33:59 GMT
```

HTML Forms

Web pages often contain fields where you can enter information. Examples include select boxes, check boxes, and fields where you can type information. Table 2.2 lists some popular HTML form tags.

TABLE 2.2 Some Common HTML Form Tags

TAG	DESCRIPTION
`<form>...</form>`	Container for the entire form
`<input />`	Data entry element; includes text, password, check box and radio button fields, and submit and reset buttons
`<select>...</select>`	Drop-down select box
`<option>...</option>`	Selectable option within select box
`<textarea>...</textarea>`	Text entry field with multiple rows

After you have completed the form you are usually invited to submit it, using an appropriately labeled button or other page element.

At this point, the HTML form constructs and sends an HTTP request from the user-entered data. The form can use either the `GET` or `POST` request type, as specified in the `method` attribute of the `<form>` tag.

`GET` and `POST` Requests

Occasionally you may hear it said that the difference between `GET` and `POST` requests is that `GET` requests are just for `GET`ting (that is, retrieving) data, whereas `POST` requests can have many uses, such as uploading data, sending mail, and so on.

Although there may be some merit in this rule of thumb, it's instructive to consider the differences between these two HTTP requests in terms of how they are constructed.

A `GET` request encodes the message it sends into a *query string*, which is appended to the URL of the server resource. A `POST` request, on the other hand, sends its message in the *message body* of the request. What actually happens at this point is that the entered data is encoded and sent, via an HTTP request, to the URL declared in the `action` attribute of the form, where the submitted data will be processed in some way.

Whether the HTTP request is of type `GET` or `POST` and the URL to which the form is sent are both determined in the HTML markup of the form. Let's look at the HTML code of a typical form:

```
<form action="http://www.sometargetdomain.com/somepage.htm" method="post">
Your Surname: <input type="text" size="50" name="surname" />
<br />
<input type="submit" value="Send" />
</form>
```

This snippet of code, when embedded in a web page, produces the simple form shown in Figure 2.4.

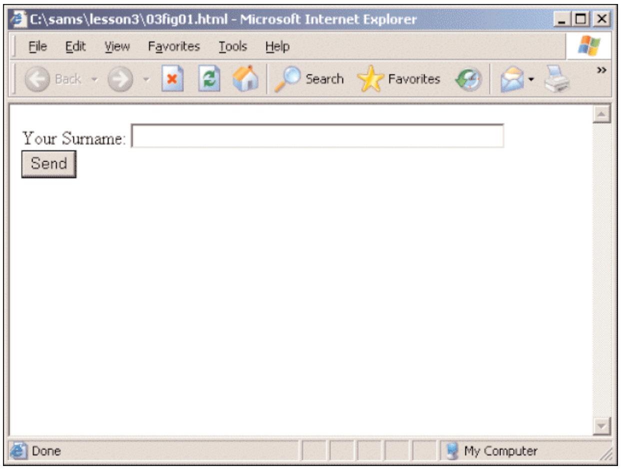

C:\sams\lesson3\03fig01.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Your Surname:
Send
Done
My Computer

FIGURE 2.4 A simple HTML form.

Let's take a look at the code, line by line. First, we begin the form by using the `<form>` tag, and in this example we give the tag two attribut-es. The `action` attribute determines the URL to which the submitted form will be sent. This may be to another page on the same server and described by a relative path, or to a remote domain, as in the code behind the form in Figure 2.4.

Next we find the attribute method, which determines whether we want the data to be submitted with a `GET` or a `POST` request.

Now suppose that we completed the form by entering the value ***Ballard*** into the `surname` field. On submitting the form by clicking the Send button, we are taken to http://www.sometargetdomain.com/somepage.htm, where the submitted data will be processed—perhaps adding the surname to a database, for example.

The variable `surname` (the `name` attribute given to the `Your Surname` input field) and its value (the data we entered in that field) will also have been sent to this destination page, encoded into the body of the `POST` request and invisible to users.

Now suppose that the first line of the form code reads as follows:

```
<form action="http://www.sometargetdomain.com/somepage.htm"
method="get">
```

On using the form, we would still be taken to the same destination, and the same variable and its value would also be transmitted. This time, however, the form would construct and send a `GET` request containing the data from the form. Looking at the address bar of the browser, after successfully submitting the form, we would find that it now contains:

```
http://www.example.com/page.htm?surname=Ballard
```

Here we can see how the parameter and its value have been appended to the URL. If the form had contained further input fields, the values entered in those fields would also have been appended to the URL as *`parameter=value`* pairs, with each pair separated by an `&` character. Here's an example in which we assume that the form has a further text input field called `firstname`:

```
http://www.example.com/page.htm?surname=Ballard&firstname=Phil
```

Some characters, such as spaces and various punctuation marks, are not allowed to be transmitted in their original form. The HTML form encodes these characters into a form that can be transmitted correctly. An equivalent process decodes these values at the receiving page before processing them, thus making the encoding/decoding operation essentially invisible to the user. We can, however, see what this encoding looks like by making a `GET` request and examining the URL constructed in doing so.

Suppose that instead of the `surname` field in our form we have a `full-name` field that asks for the full name of the user and encodes that information into a `GET` request. Then, after submitting the form, we might see the following URL in the browser:

```
http://www.example.com/page.htm?fullname=Phil+Ballard
```

Here the space in the name has been replaced by the + character; the decoding process at the receiving end removes this character and replaces the space.

The `XMLHTTPRequest` object at the heart of all Ajax applications uses HTTP to make requests of the server and receive responses. The content of these HTTP requests are essentially identical to those generated when an HTML form is submitted.

NOTE: In many cases, you may use either the `POST` or `GET` method for your form submissions and achieve essentially identical results. The difference becomes important, however, when you learn how to construct server calls in Ajax applications.

Writing Web Pages in HTML

In this section we introduce HTML, the markup language behind virtually every page of the World Wide Web. A sound knowledge of HTML provides an excellent foundation for the Ajax applications discussed in this video course.

Introducing HTML

It wouldn't be appropriate to try to give an exhaustive account of HTML (Hypertext Markup Language)—or, indeed, any of the other component technologies of Ajax—within this course. Instead we'll review the fundamental principles and give some code examples to illustrate them, paying particular attention to the subjects that will become relevant when we start to develop Ajax applications.

What Is HTML?

The World Wide Web is constructed from many millions of individual pages, and those pages are, in general, written in Hypertext Markup Language, better known as HTML.

That name gives away a lot of information about the nature of HTML. We use it to mark up our text documents so that web browsers know how to display them and to define hypertext links within them to provide navigation within or between them.

Anyone who (like me) can remember the old pre-WYSIWYG word processing programs will already be familiar with text markup. Most of these old applications required that special characters be placed at the beginning and end of sections of text that you wanted to be displayed as (for instance) bold, italic, or underlined text.

What Tools Are Needed to Write HTML?

Because the elements used in HTML markup employ only ordinary keyboard characters, all you really need is a good text editor to construct HTML pages. Many are available, and most operating systems have at least one such program already installed. If you're using some version of Windows, for example, the built-in Notepad application works just fine.

TIP: Although Notepad is a perfectly serviceable text editor, many so-called *programmers' editors* are available offering useful additional functions such as line numbering and syntax highlighting. Many of these are under open source licences and can be downloaded and used at no cost. It is well worth considering using such an editor, especially for larger or more complex programming tasks.

The use of word processing software can cause problems due to unwanted markup and other symbols that such programs often embed in the output code. If you choose to use a word processor, make sure that it is capable of saving files as plain ASCII text.

ON THE DVD: The DVD accompanying in this package contains the popular and capable jEdit programmer's editor.

Our First HTML Document

Let's jump right in and create a simple HTML document. Open your choen editor and enter the text shown in Listing 2.1. The HTML markup elements (often referred to as *tags*) are the character strings enclosed by < and >.

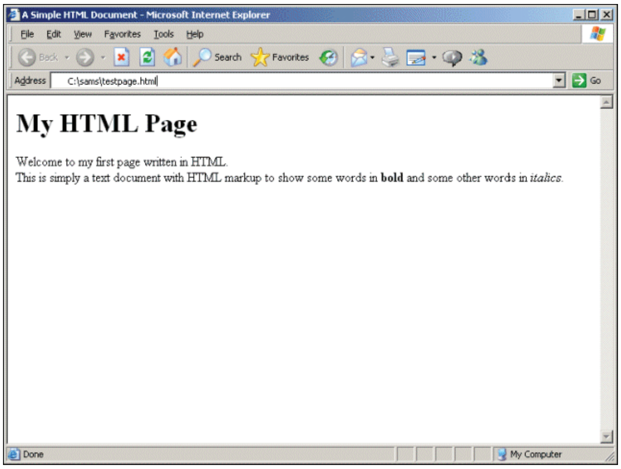
LISTING 2.1 testpage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>A Simple HTML Document</title>
</head>
<body>
<h1>My HTML Page</h1>
Welcome to my first page written in HTML.<br />
This is simply a text document with HTML markup to show some
```

```
words in <b>bold</b> and some other words in <i>italics</i>.
<br />
</body>
</html>
```

Now save the document somewhere on your computer, giving it the name testpage.html.

If you now load that page into your favorite browser, such as Internet Explorer or Firefox, you should see something like the window displayed in Figure 2.5.


A Simple HTML Document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\sams\testpage.html
Go
Done
My Computer

My HTML Page

Welcome to my first page written in HTML.
This is simply a text document with HTML markup to show some words in **bold** and some other words in *italics*.

FIGURE 2.5 Our test document displayed in Internet Explorer.

Elements of an HTML Page

Let's look at Listing 2.1 in a little more detail.

The first element on the page is known as the `DOCTYPE` element. Its purpose is to notify the browser of the "flavor" of HTML used in the document. The `DOCTYPE` element used throughout this course refers to *HTML 4.0 Transitional*, a fairly forgiving version of the HTML specification that allows the use of some earlier markup styles and structures in addition to the latest HTML 4.0 specifications.

The `DOCTYPE` element must always occur right at the beginning of the HTML document.

CAUTION: Although many modern browsers correctly display HTML without these tags, it is bad practice to omit them. Even if the page is shown correctly on your own PC, you have no idea what operating system and browser a visitor may be using—he or she may not be so lucky.

Next, note that the remainder of the document is enclosed by the elements `<html>` at the start of the page and `</html>` at the end. These tags notify the browser that what lies between should be interpreted and displayed as an HTML document.

The document within these outer tags is split into two further sections. The first is enclosed in `<head>` and `</head>` tags, and the second is contained between `<body>` and `</body>`. Essentially, the document's head section is used to store information about the document that is not to be displayed in the browser window, whereas the body of the document contains text to be interpreted and displayed to the user via the browser window.

The `<head>` of the Document

From Listing 2.1 we can see that the head section of our simple HTML document contains only one line—the words `A Simple HTML Document` enclosed in `<title>` and `</title>` tags.

Remember that the head section contains information that is not to be displayed in the browser window. This is not, then, the title displayed at the top of our page text, as you can confirm by looking again at Figure 2.5. Neither does the document title refer to the filename of the document, which in this case is testpage.html.

In fact, the document title fulfils a number of functions, among them:

- Search engines often use the page title (among other factors) to help them decide what a page is about.
- When you bookmark a page, it is generally saved by default as the document title.
- Most browsers, when minimized, display the title of the current document on their icon or taskbar button.

It's important, therefore, to choose a meaningful and descriptive title for each page that you create.

Many other element types are used in the head section of a document, including `link`, `meta`, and `script` elements. Although we don't give an account of them here, they are described throughout the course as they occur.

The Document `<body>`

Referring again to Listing 2.1, we can clearly see that the content of the document's body section is made up of the text we want to display on the page, plus some tags that help us to define how that text should look.

To define that certain words should appear in bold type, for example, we enclose those words in `<b>` and `</b>` tags. Similarly, to convert certain words into an italic typeface, we can use the `<i>` and `</i>` tags.

The heading, `My HTML Page`, is enclosed between `<h1>` and `</h1>` tags. These indicate that we intend the enclosed text to be a heading. HTML allows for six levels of headings, from `h1` (the most prominent) to `h6`. You can use any of the intermediate values `h2`, `h3`, `h4`, and `h5` to display pages having various levels of subtitles, for instance corresponding to chapter, section, and paragraph headings. Anything displayed within header tags is displayed on a line by itself.

All the tags discussed so far have been *containers*—that is, they consist of opening and closing tags between which you place the text that you want these tags to act upon. Some elements, however, are not containers but can be used alone. Listing 2.1 shows one such element: the `<br />` tag, which signifies a line break. Another example is `<hr />` (a horizontal line).

TIP: If you want to write in the body section of the HTML page but *don't* want it to be interpreted by the browser and therefore displayed on the screen, you may do so by writing it as a *comment*. HTML comments start with the character string `<!—` and end with the string `—>` as in this example:

```
<!— this is just a comment and won't be displayed in the
browser —>
```

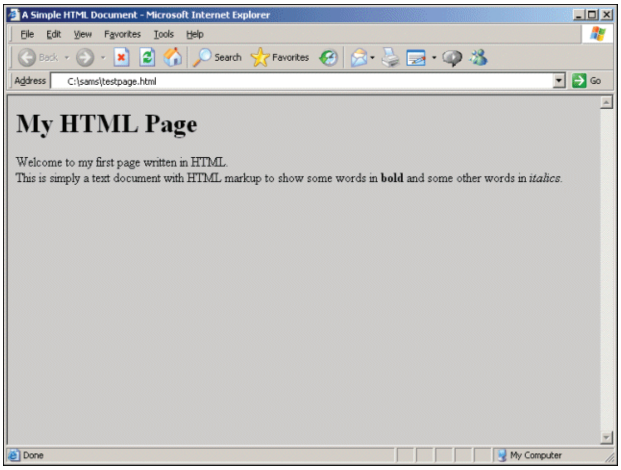
Adding Attributes to HTML Elements

Occasionally there is a need to specify exactly how a markup tag should behave. In such cases you can add (usually within the opening tag) parameter and value pairs, known as *attributes*, to change the behavior of the element:

```
<body bgcolor="#cccccc">
… page content goes here …
</body>
```

TIP: Color values in HTML are coded using a hexadecimal system. Each color value is made up from three component values, corresponding to red, green, and blue. Each of the color values can range from hex `00` to hex `ff` (zero to 255 in decimal notation). The three hex numbers are concatenated into a string prefixed with a hash character `#`. The color value `#000000` therefore corresponds to black, and `#ffffff` to pure white.

In this example, the behavior of the `<body>` tag has been modified by adjusting its `BGCOLOR` (background color) property to a light gray. Figure 2.6 shows the effect this has if 6applied to our file testpage.html:


A Simple HTML Document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\sams\testpage.html Go
Done
My Computer

My HTML Page

Welcome to my first page written in HTML.
This is simply a text document with HTML markup to show some words in **bold** and some other words in *italics*.

FIGURE 2.6 Our test page with the body color changed to gray.

Images

Images can be inserted in our page by means of the `<img />` tag. In this case we specify the source file of the image as a parameter by using the `src` attribute. Other aspects of the image display that we can alter this way include the borders, width, and height of the image:

```
<img src="myimagefile.jpg" border="2" width="250" height="175" />
```

Border width, image width, and image height are in numbers of *pixels* (the "dots" formed by individual picture elements on the screen).

TIP: A further useful attribute for images is `alt`, which is an abbreviation of *alternative text*. This specifies a short description of the image that will be offered to users whose browsers cannot, or are configured not to, display images. Alternative text can also be important in making your website accessible to those with visual impairment and other disabilities:

```
<img src="myimagefile.jpg" alt="Description of Image" />
```

Tables

Often you want to display information in tabular format, and HTML has a set of elements designed specifically for this purpose:

```
<table>
<tr><th>Column Header 1</th><th>Column Header 2</th></tr>
<tr><td>Data Cell 1</td><td>Data Cell 2</td></tr>
<tr><td>Data Cell 3</td><td>Data Cell 4</td></tr>
</table>
```

The `<table>` and `</table>` tags contain a nested hierarchy of other tags, including `<tr>` and `</tr>`, which define individual table rows; `<th>` and `</th>`, which indicate cells in the table's header; and `<td>` and `</td>`, which contain individual cells of table data.

Look ahead to Figure 2.7 to see an example of how a table looks when displayed in a browser window.

Hyperlinks

Hypertext links (*hyperlinks*) are fundamental to the operation of HTML. By clicking on a hyperlink, you can navigate to a new location, be that to another point on the current page or to some point on a different page on another website entirely.

Links are contained within an `<a>`, or anchor tag, a container tag that encloses the content that will become the link. The destination of the link is passed to this tag as a parameter `href`:

```
Here is <a href="newpage.html">my hyperlink</a>
```

Clicking on the words `my hyperlink` in the above example results in the browser requesting the page newpage.html.

TIP: A hyperlink can contain images as well as, or instead of, text. Look at this example:

```
<a href="newpage.html"><img src="picfile.gif" /></a>
```

Here, a user can click on the image picfile.gif to navigate to newpage.html.

A More Advanced HTML Page

Let's revisit our testpage.html and add some extra elements. Listing 2.2 shows seville.html, developed from our original HTML page but with different content in the `<body>` section of the document. Figure 2.7 shows how the page looks when displayed, this time in Mozilla Firefox.

Now we have applied a background tint to the body area of the document. The content of the body area has been centered on the page, and that content now includes an image (which we've given a two-pixel-wide border), a heading and a subheading, a simple table, and some text.

LISTING 2.2 seville.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>A Simple HTML Document</title>
</head>
<body bgcolor="#cccccc">
<center>
<img src="cathedral.jpg" border="2" alt="Cathedral" />
<h1>Guide to Seville</h1>
<h3>A brief guide to the attractions</h3>
<table border="2">
<tr>
  <th bgcolor="#aaaaaa">Attraction</th>
  <th bgcolor="#aaaaaa">Description</th>
</tr>
<tr>
  <td>Cathedral</td>
  <td>Dating back to the 15th century</td>
</tr>
<tr>
  <td>Alcazar</td>
  <td>The medieval Islamic palace</td>
</tr>
</table>
<p>Enjoy your stay in beautiful Seville.</p>
</center>
</body>
</html>
```

Let's take a closer look at some of the code.

First, we used the `BGCOLOR` property of the `<body>` tag to provide the overall background tint for the page:

```
<body bgcolor="#cccccc">
```

Everything in the body area is contained between the `<center>` tag (immediately after the body tag) and its partner `</center>`, immediately before the closing body tag. This ensures that all of our content is centered on the page.

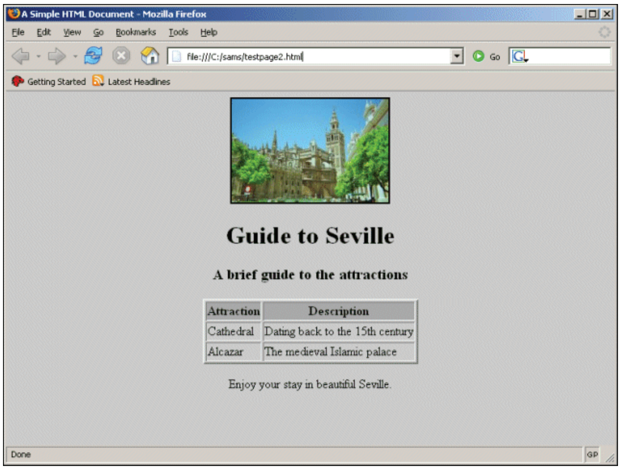
The main heading is enclosed in `<h1>` … `</h1>` tags as previously, but is now followed by a subheading using `<h3>` … `</h3>` tags to provide a slightly smaller font size.

By using the border property in our opening `<table>` tag, we set a border width of two pixels for the table:

```
<table border="2">
```

Meanwhile we darkened the background of the table's header cells slightly by using the `BGCOLOR` property of the `<th>` elements:

```
<th bgcolor="#aaaaaa">Vegetables</th>
```


A Simple HTML Document - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///C:/sams/testpage2.html
Go
Getting Started
Latest Headlines
Done

Guide to Seville

A brief guide to the attractions

Attraction	Description
Cathedral	Dating back to the 15th century
Alcazar	The medieval Islamic palace

Enjoy your stay in beautiful Seville.

FIGURE 2.7 testpage2.html shown in Mozilla Firefox.

Some Useful HTML Tags

Table 2.3 lists some of the more popular HTML tags.

TABLE 2.3 Some Common HTML Markup Elements

DOCUMENT TAGS	
`<html>..</html>`	The entire document
`<head>..</head>`	Document head
`<body>..</body>`	Document body
`<title>..</title>`	Document title
STYLE TAGS	
`<a>..</a>`	Hyperlink
`<b>..</b>`	Bold text
`<em>..</em>`	Emphasized text
`<font>..</font>`	Changed font
`<i>..</i>`	Italic text
`<small>..</small>`	Small text
`<table>..</table>`	Table
`<tr>..</tr>`	Table row
`<th>..</th>`	Cell in table header
`<td>..</td>`	Cell in table body
`<ul>..</ul>`	Bulleted list
`<ol>..</ol>`	Ordered (numbered) list
`<li>..</li>`	List item in bulleted or ordered list

Adding Your Own Style

TIP: The World Wide Web Consortium is responsible for administering the definitions of HTML, HTTP, XML, and many other web technologies. Its website is at http://www.w3.org/.

As you've already learned, HTML was written as a markup language for defining the structure of a document (paragraphs, headings, tables, and so on). Although it was never intended to become a desktop publishing tool, it does include some basic formatting attributes, such as `font-size`, `align` and the aforementioned `bgcolor`. In 1996, the W3C first recommended the idea of Cascading Style Sheets (CSS) to format HTML documents. The recommendation, which was updated in mid-1998, enables Web developers to separate the structure and format of their documents.

The CSS recommendation describes the following three types of style sheets:

- **Embedded** The style properties are included (within the `<style>` tags) at the top of the HTML document. A style assigned to a particular tag applies to all those tags in this type of document. In this course, you'll see embedded style sheets most often.
- **Inline** The style properties are included throughout the HTML page. Each HTML tag receives its own style attributes as they occur in the page.
- **Linked** The style properties are stored in a separate file. That file can be linked to any HTML document using a `<link>` tag placed within the `<head>` tags.

In the following sections, you'll learn how to construct these style sheets and how to apply them to your documents.

TIP: Even without all the formatting benefits that style sheets provide, Web developers can rejoice in knowing that using style sheets will no doubt be the biggest timesaver they've ever encountered. Because you can apply style sheets to as many HTML documents as you like, making changes takes a matter of minutes rather than days.

Before the advent of style sheets, if you wanted to change the appearance of a particular tag in your Web site, you would have to open each document, find the tag you wanted to change, make the change, save the document, and continue on to the next document. With style sheets, you can change the tag in a single style sheet document and have the changes take effect immediately in all the pages linked to it.

Defining the Rules

Style sheet rules are made up of selectors (the HTML tags that receive the style) and declarations (the style sheet properties and their values). In the following example, the selector is the `body` tag and the declaration is made up of the style property (`background`) and its value (`black`). This example sets the background color for the entire document to black.

```
body {background:black}
```

You can see that, in a style sheet, the HTML tag is not surrounded by brackets as it would be in the HTML document, and the declaration is surrounded by curly braces. Declarations can contain more than one property. The following example also sets the text color for this page to white. Notice that the two properties are separated by a semicolon.

```
body {background:black; color:white}
```

If you want to apply the same rules to several HTML tags, you could group those rules together, as in the following example:

```
body, td, h1 {
              background:black;
              color:white
              }
```

Add a Little `class`

As the old saying goes, rules are made to be broken. What if you don't want every single `h1` heading in your document to be white on a black background? Maybe you want every other `h1` heading to be yellow on a white background. Let me introduce you to the `class` attribute. You can apply this attribute to almost every HTML tag, and it's almost like creating your own tags.

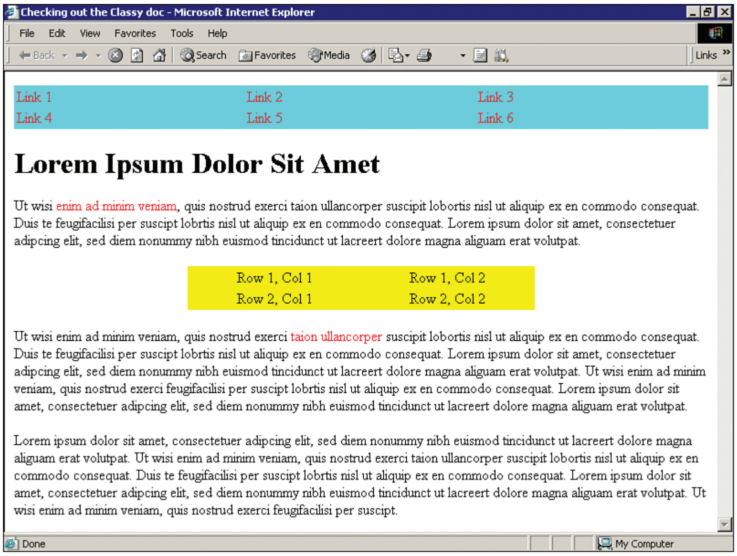
Figure 2.8 shows a fairly standard HTML page that uses an aqua table at the top of the page to hold the navigation links, and places other tabular content in yellow tables throughout the document. You can see the HTML document for that page in Figure 2.9.

Take a closer look at the style properties in Figure 2.9. This document defines two `table` styles within the `<style>` tags. The HTML tag name `table` is followed by a period (`.`) and the `class` names (`nav` and `rest`).

```
table.nav {background:aqua}
table.rest {background:yellow;
            text-align:center;
            color:black}
```

When the table is referenced in the body of the document, you must apply the `class` attribute to tell the browser which style properties should be applied. The HTML markup for each table in this example appears in the following HTML code. You can see that the `class` name appears within quotations just like the other HTML attributes (and as with the `width` attribute shown here).

```
<table class="nav" width="100%">
<table class="rest" width=50%>
```


Checking out the Classy doc - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links
Done
My Computer

Link 1	Link 2	Link 3
Link 4	Link 5	Link 6

Lorem Ipsum Dolor Sit Amet

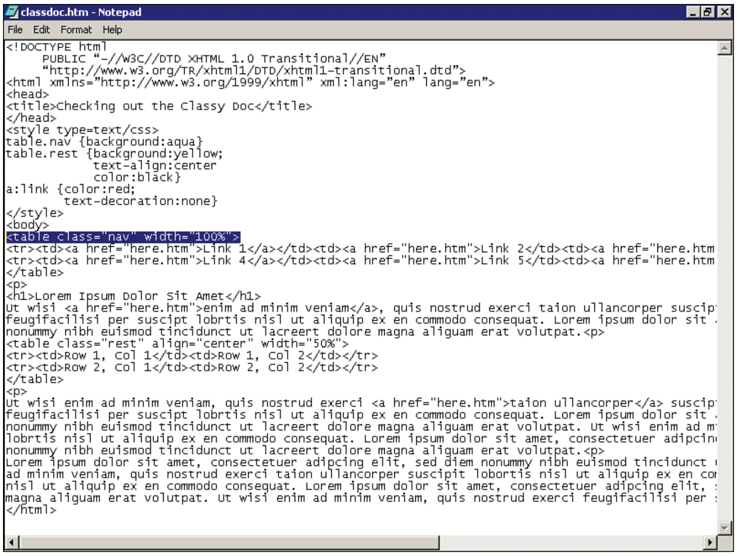
Ut wisi enim ad minim veniam, quis nostrud exerci taion ullancorper suscipit lobortis nisl ut aliquip ex en commodo consequat. Duis te feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat.

Row 1, Col 1	Row 1, Col 2
Row 2, Col 1	Row 2, Col 2

Ut wisi enim ad minim veniam, quis nostrud exerci taion ullancorper suscipit lobortis nisl ut aliquip ex en commodo consequat. Duis te feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat.

Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci taion ullancorper suscipit lobortis nisl ut aliquip ex en commodo consequat. Duis te feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci feugifacilisi per suscipt.

FIGURE 2.8 An HTML page that formats two tables differently.



```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Checking out the Classy Doc</title>
</head>
<style type=text/css>
table.nav {background:aqua}
table.rest {background:yellow;
            text-align:center
            color:black}
a:link {color:red;
        text-decoration:none}
</style>
<body>
<table class="nav" width="100%">
<tr><td><a href="here.htm">Link 1</a></td><td><a href="here.htm">Link 2</td><td><a href="here.htm
<tr><td><a href="here.htm">Link 4</a></td><td><a href="here.htm">Link 5</td><td><a href="here.htm
</table>
<p>
<h1>Lorem Ipsum Dolor Sit Amet</h1>
Ut wisi <a href="here.htm">enim ad minim veniam</a>, quis nostrud exerci taion ullancorper suscip
feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit
nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat.<p>
<table class="rest" align="center" width="50%">
<tr><td>Row 1, Col 1</td><td>Row 1, Col 2</td></tr>
<tr><td>Row 2, Col 1</td><td>Row 2, Col 2</td></tr>
</table>
<p>
Ut wisi enim ad minim veniam, quis nostrud exerci <a href="here.htm">taion ullancorper</a> suscip
feugifacilisi per suscipt lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit
nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat. Ut wisi enim ad m
lobrtis nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcin
nonummy nibh euismod tincidunct ut lacreert dolore magna aliguam erat volutpat.<p>
Lorem ipsum dolor sit amet, consectetuer adipcing elit, sed diem nonummy nibh euismod tincidunct
ad minim veniam, quis nostrud exerci taion ullancorper suscipit lobortis nisl ut aliquip ex en co
nisl ut aliquip ex en commodo consequat. Lorem ipsum dolor sit amet, consectetuer adipcing elit,
magna aliguam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci feugifacilisi per
</html>
```

FIGURE 2.9 The HTML document for the page in Figure 2.8. Notice the class attribute in each `<table>` tag.

Applying Styles

Before moving on, we'll quickly cover how to apply style properties to your documents. Remember, you have three methods to add style sheets: embedded, linked, and inline. We'll discuss each one in turn.

Embedded Styles

All the styles are defined at the top of the HTML document within the `<head>` tags because they contain information about the entire document. The styles defined here apply only to the one document in which they appear. If you plan to use these same styles in another document, you need to add them there as well.

NOTE: The `<style>` tag almost always includes the `type="text/css"` attribute, so you should get used to adding it.

```
<head>
<style type="text/css">
table.nav {background:aqua}
table.rest {background:yellow;
            text-align:center;
            color:black}
a:link {color:red;
        text-decoration:none}
</style>
</head>
```

Linked Styles

Linked style sheets hold all the style properties in a separate file. You then link the file into each HTML document where you want those style properties to appear.

```
<head>
<link rel="stylesheet" href="mystyles.css" type="text/css">
</head>
```

With this method, I've created a separate file called mystyles.css (for cascading style sheet) that contains all my style properties. You can see that the same `type="text/css"` attribute shows up here. Following are the entire contents of the mystyles.css file. These are the same styles that showed up in the preceding embedded styles example, but now they appear in a separate text file.

```
table.nav {background:aqua}
table.rest {background:yellow;
            text-align:center;
            color:black}
a:link {color:red;
        text-decoration:none}
```

Inline Styles

With inline styles, the style properties are added to the HTML tag as the tag is entered. This means that if I want the same style to appear on all the `<h1>` tags in my document, I would have to type those styles in all the `<h1>` tags. Look at the following example. I am still using the same style properties, as in the previous examples, but now you can see how the two tables would be created using inline styles.

```
<table style="background:aqua" width="100%">

<table style="background:yellow; text-align:center;
             color:black" width="100%">
```

Using inline styles, the `<style>` tag becomes the `style` attribute. Multiple style properties are still separated by semicolons, but the entire group of properties for each tag is grouped within each HTML tag. This type of style sheet is fine for documents in which you need to apply styles to only one or two elements, but you wouldn't want to do all that work when you have a lot of styles to add.

Cascading Precedence

Web browsers give precedence to the style that appears closest to the tag. So, inline styles (which appear as attributes within the tag itself) are most important. Embedded styles (which appear at the top of the HTML file) are applied next, and linked styles (which appear in another file altogether) are applied last.

Imagine that you have created an embedded style for the `<h1>` tag, but want to change that style for one occurrence of the `<h1>` tag in that document. You would create an inline style for that new `<h1>` tag. The browsers recognize that fact and change the style for that tag to reflect the inline style.

CAUTION: Style sheet precedence is supposed to place more importance on embedded styles than on linked style sheets. In actual practice, however, you'll find that both Internet Explorer and Netscape treat linked sheets as more important than embedded sheets (but they do treat inline styles as more important than either of the other two). You'll find that you have better luck if you use either linked or embedded styles, but not both.

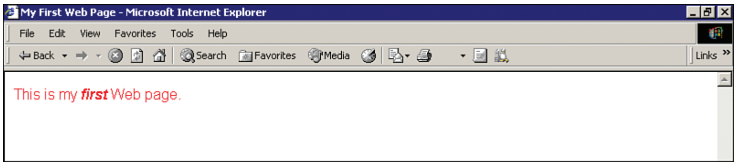
Formatting Text with Styles

Text is the most important element of any Web page. Without text, there is nothing on the page to help people decide whether it's worth coming back.

Text on an HTML page is structured by the `<body>`, `<p>`, `<td>`, `<tr>`, `<th>`, `<h1>` `<h6>`, and `<li>` tags (among others). You can add your own style preferences to each of these tags using the style properties shown in Table 2.4.

In the following example, we've added some embedded style elements that set the font, font size, and font color for the body text of a basic HTML page. In Figure 2.10, you can see how those styles change the appearance of the document in the browser.

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
<title>My First Web Page</title>
<style type="text/css">
body {font-family:"Arial";
      font-size:"12pt";
      color:red}
</style>
</head>
<body>
<p>This is my <b><i>first</i></b> Web page.</p>
</body>
</html>
```


My First Web Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links

This is my ***first*** Web page.

FIGURE 2.10 The browser applies the style attributes to the text in the `<body>` tags.

Table 2.4 lists the many style properties that you can use to format your text.

TABLE 2.4 Style Properties for Text

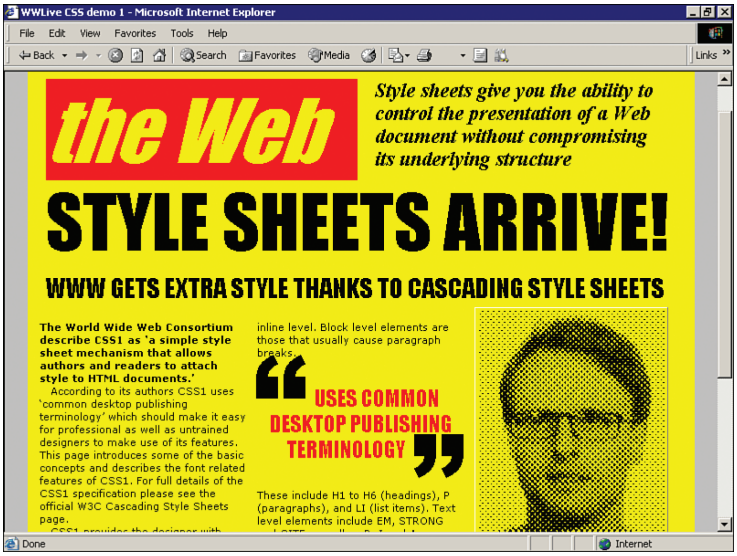
PROPERTY	DESCRIPTION OF USE AND VALUES
`background`	Sets the background color for the text.
`color`	Sets the text color for the text.
`font-family`	Sets the font for the text.
`font-size`	Can be a point size, a percentage of the size of another tag, or `xx-small` to `xx-large`.
`font-style`	`normal` (which is assumed) or `italic`.
`font-weight`	`extra-light` to `extra-bold`.
`text-align`	`left`, `right`, `center`, or `justify` (full).
`text-indent`	Can be a fixed length or a percentage.
`text-decoration`	`underline`, `overline`, `strikethrough`, and `none.`

Microsoft maintains a brief tutorial for style sheets on its typography site (`http://www.microsoft.com/typography/default.mspx`). The tutorial teaches Web page authors how style sheets can enhance their documents. The `<style>` tag for one of those examples is shown in the following code. This is impressive because of the many different styles and classes defined in this document. You can see that you are only limited by your own imagination. You can see the page this style code created in Figure 2.11.

```
<style type="text/css">
body {background: coral}
.copy {color: Black;
    font-size: 11px;
    line-height: 14px;
    font-family: Verdana, Arial, Helvetica, sans-serif}
a:link {text-decoration: none;
    font-size: 20px;
    color: black;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
.star {color: white;
    font-size: 350px;
    font-family: Arial, Arial, helvetica, sans-serif}
.subhead {color: black;
    font-size: 28px;
    margin-top: 12px;
    margin-left: 20px;
    line-height: 32px;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
```

```
.what {color: black;
    font-size: 22px;
    margin-left: 20px;
    font-weight: bold;
    font-style: italic;
    font-family: Times New Roman, times, serif}
.quott {color: black;
    font-size: 120px;
    line-height: 120px;
    margin-top: -24px;
    margin-left: -4px;
    font-family: Arial Black, Arial, helvetica, sans-serif}
.quotb {color: black;
    font-size: 120px;
    line-height: 120px;
    margin-right: -1px;
    margin-top: -33px;
    font-family: Arial Black, Arial, helvetica, sans-serif}
.quote {color: red;
    font-size: 24px;
    line-height: 28px;
    margin-top: -153px;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
.footer {color: cornsilk;
    background: red;
    font-size: 22px;
    margin-left: 20px;
    margin-top: 16px;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
.headline {color: black;
    font-size: 80px;
    line-height: 90px;
    margin-left: 20px;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
.mast {color: cornsilk;
    font-size: 90px;
    font-style: italic;
    font-family: Impact, Arial Black, Arial,
                 Helvetica, sans-serif}
</style>
```

CAUTION: None of the most popular Web browsers react the same way to all the style sheet properties. Your best bet is to remember to test everything before you publish it. Webmaster Stop maintains a table of style sheet properties mapped to the most popular browsers. Check out this table (`http://www.webmasterstop.com/118.html`) to find out whether the style sheet properties you plan to use are supported by specific browsers.


WWLive CSS demo 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links
Done
Internet

the Web

Style sheets give you the ability to control the presentation of a Web document without compromising its underlying structure

STYLE SHEETS ARRIVE!

WWW GETS EXTRA STYLE THANKS TO CASCADING STYLE SHEETS

The World Wide Web Consortium describe CSS1 as 'a simple style sheet mechanism that allows authors and readers to attach style to HTML documents.'

According to its authors CSS1 uses 'common desktop publishing terminology' which should make it easy for professional as well as untrained designers to make use of its features. This page introduces some of the basic concepts and describes the font related features of CSS1. For full details of the CSS1 specification please see the official W3C Cascading Style Sheets page.

inline level. Block level elements are those that usually cause paragraph breaks.

“USES COMMON DESKTOP PUBLISHING TERMINOLOGY”

These include H1 to H6 (headings), P (paragraphs), and LI (list items). Text level elements include EM, STRONG

FIGURE 2.11 The preceding style code produced this page, found at http://www.microsoft.com/typography/css/gallery/slide3.htm.

Link Styles

You have probably seen those bright blue underlined hyperlinks on the Web. Style sheets have the following selectors to help you change the look of them:

- `a:link` Sets the styles for unvisited links.
- `a:visited` Sets the styles for visited links.
- `a:active` Sets the styles for the link while it is linking.
- `a:hover` Sets the style for the link while your mouse is hovering.

Table 2.5 shows some of the style properties you can assign to your links.

Table 2.5 Style Properties for the Anchor Styles

PROPERTY	DESCRIPTION OF USE AND VALUES
`background-color`	Sets the background color for the link.
`color`	Sets the text color for the link.
`font-family`	Sets the font for the text of the link.
`text-decoration`	`underline`, `overline`, `strikethrough`, and `none`.

TIP: One of the most popular style sheet effects on the Web right now is to remove the underlining on hyperlinks. To do this on your pages, just add the `text-decoration:none` declaration to the `a` styles, as shown in the following example:

```
a:link {color:yellow;
        text-decoration:none}
```

If you like the look of the underlined hyperlink, you're in luck. You don't have to specify anything at all. Underlining is assumed for all `a` styles.

CAUTION: Don't forget to test your pages before you publish them. Not all colors work together. If you've specified a black background color and a black text color, you've got a problem because no one will be able to see your text.

Color Styles

As you can see in Table 2.6, you can apply color to your HTML tags in two different ways: with `color` or with `background`.

TABLE 2.6 Style Properties for Color

PROPERTY	DESCRIPTION OF USE AND VALUES
`color`	Sets the color of the text.
`background`	Sets the background of the page or text.

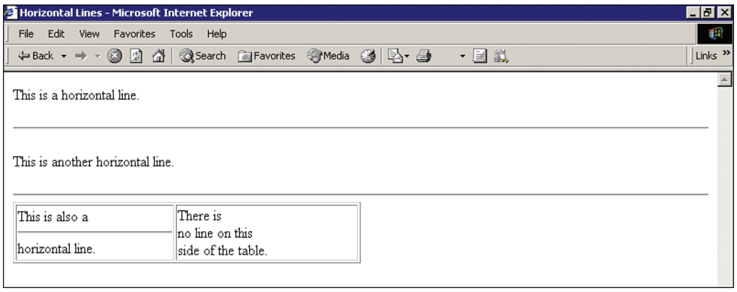
Adding Lines

A horizontal line, or horizontal rule as' it is named in HTML, is one of the easiest tags to use. You can insert the `<hr />` tag anywhere in your document to insert a horizontal line that extends across the space available. Take a look at the following sample HTML. It shows three `<hr>` tags: two used as a section break between text and the other used inside a table cell. Figure 2.12 shows how they appear in the browser.

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
<title>Horizontal Lines</title>
<style type="text/css">
td {text-align=center}
</style>
</head>
<body>
<p>This is a horizontal line.</p>
<hr />
```

```
<p>This is another horizontal line.</p>
<hr />

<table width="50%" rules=cols>
  <tr>
   <td>This is also a<hr />horizontal line.</td>
   <td>There is <br />no line on this<br />side
       of the table.</td>
  </tr>
</table>
</body>
</html>'
```


Horizontal Lines - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Links

This is a horizontal line.

This is another horizontal line.

This is also a horizontal line.	There is no line on this side of the table.

FIGURE 2.12 The `<hr />` tag inserts a horizontal line that stretches across the available horizontal space.

Margin Styles

Style sheets give you another important' advantage: You can specify the margins of almost any HTML element. The margins can be defined in `pt`, `in`, `cm`, or `px` sizes.

```
body {margin-left: 100px;
     margin-right: 100px;
     margin-top: 50px}
```

You can set the `margin-left`, `margin-right`, and `margin-top` properties individually or combine them into one property called `margin` that applies the sizes to the `top`, `right`, and `left` margins.

```
body {margin: 100px 100px 50px}
```

Anatomy of an Ajax Application

In this section you'll learn about the individual building blocks of Ajax and how they fit together to form the architecture of an Ajax application.

The Need for Ajax

In other parts of the course, 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 2.13 illustrates this interaction.

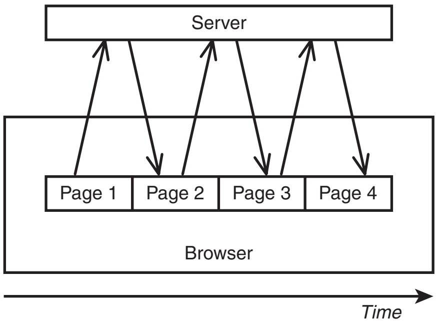


FIGURE 2.13 Traditional client–server interactions.

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 press the Submit button a second time," and "don't press 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 2.14 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.

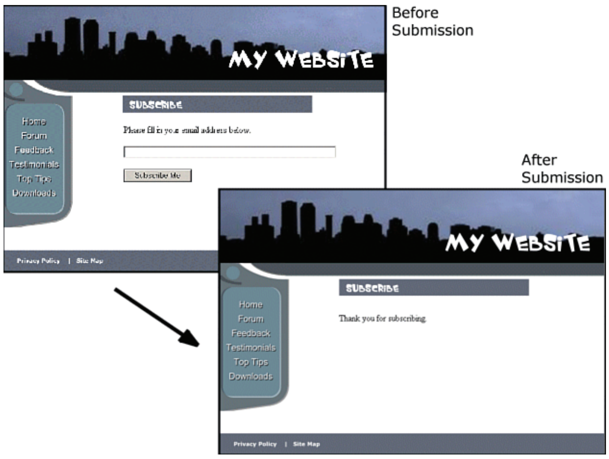


FIGURE 2.14 Many page items are reloaded unnecessarily.

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

NOTE: *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).

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 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 2.15 represents how these interactions take place.

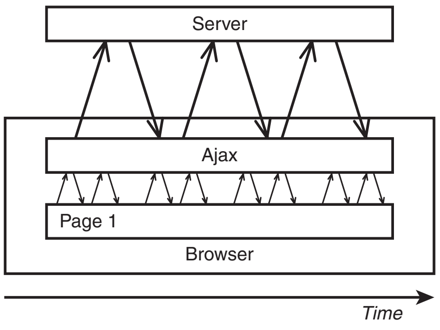


FIGURE 2.15 Ajax client–server interaction.

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 2.16 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.

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

NOTE: 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.

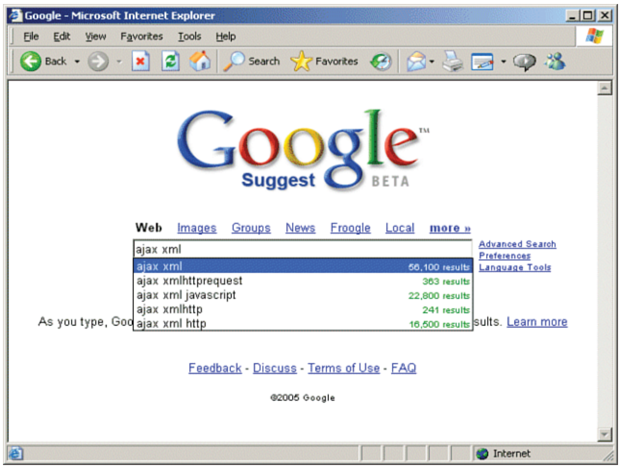

Google - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Google Suggest BETA
Web Images Groups News Froogle Local more »
Advanced Search
Preferences
Language Tools
ajax xml
ajax xml 56,100 results
ajax xmlhttprequest 363 results
ajax xml javascript 22,800 results
ajax xmlhttp 241 results
ajax xml http 16,500 results
As you type, Goo sults. Learn more
Feedback - Discuss - Terms of Use - FAQ
©2005 Google
Internet

FIGURE 2.16 An example of an Ajax application—Google Suggest.

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.

TIP: 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.

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.

Later on 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.

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.

NOTE: In this course 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.

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 on we'll look at some examples of calling web services using protocols such as SOAP and REST.

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.

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.

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. 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 2.17 shows the flow diagram of all this.

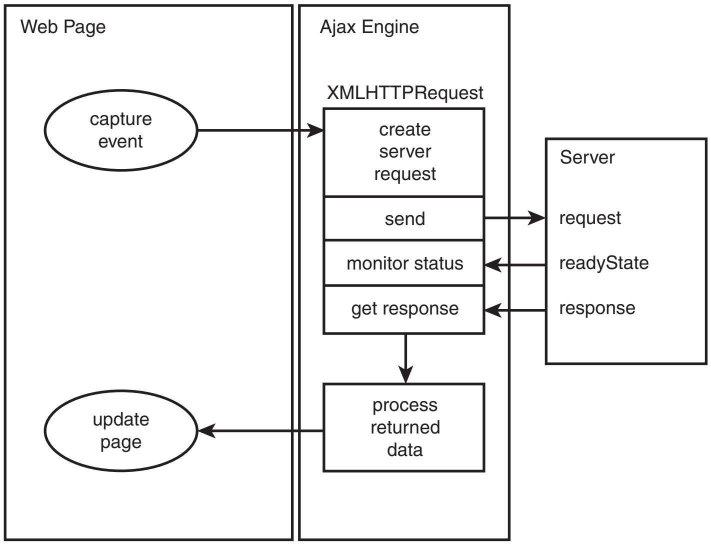
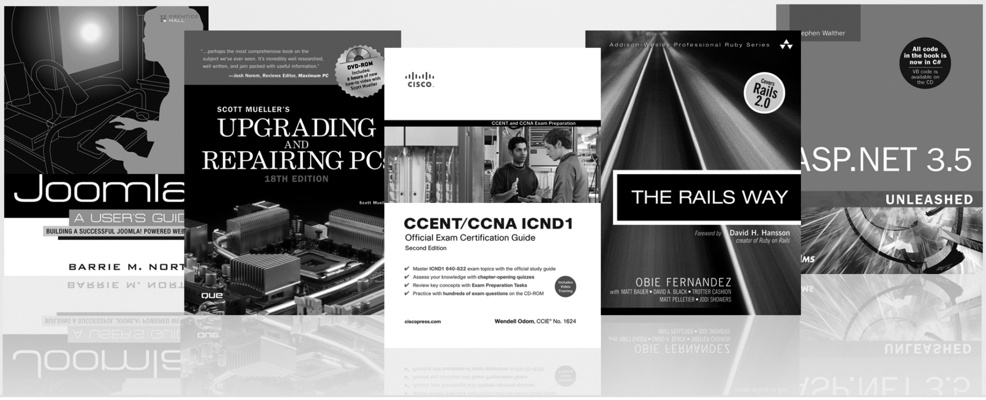


FIGURE 2.17 How the components of an Ajax application work together.

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 re-used 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".


Joomla! A User's Guide
Barrie M. North
Scott Mueller's Upgrading and Repairing PCs 18th Edition
CCENT/CCNA ICND1 Official Exam Certification Guide Second Edition
Wendell Odom
The Rails Way
Obie Fernandez
ASP.NET 3.5 Unleashed