Making Everything Easier!™

HTML5 Programming with JavaScript®

FOR DUMMIES®

A Wiley Brand

Learn to:

- Build dynamic pages using JavaScript with HTML
- Create websites with greater interactivity
- Use animation, libraries, the canvas element, and more
- Harness the power of HTML5

John Paul Mueller
Coauthor of the bestselling
C++ All-in-One For Dummies

Chapter 1

HTML, Say Hello to JavaScript

In This Chapter

- Understanding what JavaScript is
- Understanding how JavaScript fits in with HTML5

JavaScript is a text-based scripting language that's interpreted by a client system to perform tasks in various settings. The most common setting is within browsers. A developer wants to do something special, such as accept input from a form, and JavaScript makes it possible.

JavaScript appears in many other places. For example, Windows has long allowed the use of JavaScript to create applications, and now it has an even bigger role with Windows 8. (See `http://msdn.microsoft.com/library/windows/apps/br211385.aspx` for details.) Special versions of JavaScript also support application development on the Macintosh. (See `www.latenightsw.com/freeware/JavaScriptOSA` as an example.) In fact, you can even run Linux in a browser by using a JavaScript emulator. (See `www.webmonkey.com/2011/05/yes-virginia-that-is-linux-running-on-javascript` for details.) The point is that JavaScript is a language that appears in all sorts of places on many different operating systems. When you discover JavaScript, you open an exciting new world of programming that works on myriad platforms — a dream that developers have had for a very long time.

This book doesn't explore all of the possible environments in which you can use JavaScript. I doubt very much that you could examine the topic in any detail with an entire shelf of books. What you'll encounter is how JavaScript is used with HTML5, the newest version of the HyperText Markup Language (HTML). HTML5 and JavaScript are made for each other. By combining these two languages, you create a robust environment for Web applications. Modern Web applications can perform an amazing array of tasks —

everything from word processing to database entry. The use of HTML5 and JavaScript together makes it possible for anyone or any organization to move applications from the desktop to the *cloud* (a special location on the Internet used to store applications and data), where any device capable of running JavaScript can access and use them. In short, combining HTML5 with JavaScript can free users from using a specific device to interact with any application you can imagine.

Of course, any book on a programming language must begin with some basics and present some ground rules, which is precisely what you find in this chapter. You discover a little more about what JavaScript is and how it can help you create interesting applications. You'll also begin creating some basic JavaScript applications in this chapter. They won't do too much at first; you'll gain a sense of what JavaScript can do after you've worked with it some more.

JavaScript can work on any platform that supports it and in any browser that supports HTML5. To see what level of support your browser provides, go to `http://html5test.com` and enable JavaScript support (if asked). This site tells what your browser can and can't do with JavaScript so that you know whether your browser can use specific features in this book. (You may want to print the results so that you have a reference to them as you progress through the book.) For the purpose of making things easier for everyone, the scripts in this book were tested with the latest version of Firefox available at the time of writing on a Windows 7 system. (See Chapter 2 for more on the many benefits to using Firefox for developing browser-based applications.) You may see slight variations in screen output and functionality when you use a different browser or operating system.

Introducing JavaScript

Originally, the Internet allowed only *static pages* — pages that presented fixed content that couldn't change. Yes, there were links and other features that let you move to other pages, but the content on them didn't change. JavaScript was originally conceived as a means for making Web pages *dynamic* — making it possible for users to interact with them and receive something in return. In fact, that's the basic idea behind JavaScript today, but the interactions have become complex enough that you can call them applications. The following sections introduce you to what JavaScript is all about and why you need to add this language to your programming toolbox.

Discovering the history of JavaScript

You won't find a complete history of JavaScript in this book because so many people have already written about it. There are many histories of JavaScript on the Internet. One of the better histories is at `http://javascript.about.com/od/reference/a/history.htm`. This short history will provide you with a good overview of the most important facts about the creation of JavaScript. You can find a more detailed history of JavaScript at `http://www.howtocreate.co.uk/jshistory.html`.

Java and JavaScript aren't long-lost relatives

Some programmers have confused Java and JavaScript over the years, partly because of the naming similarities. It turns out that JavaScript was originally named LiveScript. Netscape saw how popular Java had become and decided to rename LiveScript to JavaScript to play off that popularity. In reality, Java and JavaScript are completely different languages, and you shouldn't confuse the two. There's nothing similar between Java and JavaScript. For example, whereas Java is a *compiled* language (one that's turned into a native executable using a special application) that requires a plug-in to run in your browser, JavaScript is an *interpreted* language (a text description of what to do that requires an interpreter, another sort of special application, to execute) that requires no special plug-in because the browser provides native support for it.

There's nothing unusual about the similarity in naming between Java and JavaScript. Vendors have used naming similarities for many products in order to obtain some sort of benefit from the popularity of similarly named products. The most important thing to remember is that you can't use any Java functionality, documentation, or tools to create your JavaScript applications. The two languages are quite different.

Recognizing the benefits of JavaScript

JavaScript is an amazing language that can perform a wide variety of tasks when you know how to use it. In fact, in many respects, JavaScript is unique in the programming world because you don't have to perform any special

tricks to get it to work in most environments. Not every environment is completely compatible with JavaScript, but you can usually get essential features of an application to work no matter which environment runs the application you create. With this in mind, you want to know what JavaScript can do for you as a developer because having this information makes it easier for you to convince management and other developers to work with you on JavaScript solutions.

The following sections discuss the most commonly cited benefits of JavaScript (although you'll almost certainly find other benefits in online articles such as the ones described at `http://ezinearticles.com/?What-Are-the-Benefits-of-JavaScript?&id=4743036`).

Using JavaScript in any browser

JavaScript is quite flexible because it's an interpreted language. Interpreted languages are distributed as plain text. Every computer platform ever created can understand plain text. Even old mainframes can understand plain text, at some level, which means that plain text is the most common form of computer communication ever created. A special browser feature, the *interpreter,* reads the text description of what to do and then performs those tasks within the browser environment. Every browser has this special feature built-in so you never need to download a special plug-in when working with JavaScript — the support you need is already available. Because the JavaScript language is essentially the same in every browser, the same text description of what tasks to perform works everywhere. This text description is the JavaScript language that you use throughout the book to create the example applications.

It's important to realize that the browser's interpreter must recognize all of the JavaScript key words and programming constructs. As JavaScript has improved, it has added new features that older interpreters don't understand. Consequently, you can't expect a really old interpreter to completely understand a JavaScript application that uses all of the latest features. In many cases, the application may still run, but with reduced functionality. In other cases, the program may crash simply because the interpreter doesn't know what to do. This is why you need to know which platforms support the combination of HTML and JavaScript you want to use in your Web pages and why you need to test your browser at `http://html5test.com` to ensure the examples in this book will work for you.

Using JavaScript with any operating system

Many programming languages rely on special operating system features. Native code programs — those that speak the operating system's special language — are especially attached to a particular operating system because the language relies on the special operating system features. JavaScript has no such reliance. All JavaScript cares about is the browser in which it runs. The browser interacts with the operating system and takes care of all of those low-level tasks so JavaScript can be generalized to work with any operating system.

Using JavaScript with any device

Some developers are used to the idea that their applications will work only on certain devices. In fact, most developers are happy when they can get an application to work on just one *platform* (the combination of a specific device matched with a specific operating system) successfully. JavaScript has no such limits.

If you have a device that has an HTML5-compatible browser with JavaScript support, it's quite likely that the applications in this book will work. (That said, I've tested the applications only on the systems specified in the book's introduction, so your results will vary depending on device and browser compatibility.) Even mobile devices will use JavaScript without problems. For example, if you have a Blackberry, it's quite likely that the examples in this book will work on it without problem. (See `http://www.sencha.com/blog/html5-scorecard-rim-blackberry-playbook-2` for details on Blackberry support for HTML5.)

Most developers will find it quite amazing that the application created with JavaScript could potentially work on platforms that didn't even exist at the time the application was written! The idea that JavaScript is everywhere will surprise many people. Don't be surprised when the Android user sitting next to you in the doctor's office is using the application you wrote in JavaScript for the PC. JavaScript doesn't care where it runs.

Accessing common platform features

As previously mentioned in this chapter, JavaScript requires an interpreter, and that interpreter translates JavaScript key words into something the underlying platform can understand. Unlike some other languages, JavaScript doesn't exist within a *sandbox* — a special programming environment that limits access to operating system features to reduce potential security problems. This means that JavaScript can tell the interpreter that it wants to save a file somewhere, and then the interpreter will do its best to satisfy that need using platform-specific functionality. JavaScript doesn't care how or where the file is saved — it simply cares that the file is saved. In short, JavaScript insulates your application from the platform in a way that makes it possible to create truly amazing applications.

There's never a free lunch when it comes to applications, however. JavaScript can perform simple tasks, such as saving a file. The catch is that you can't depend on it to use unique operating system features. For example, Windows supports file encryption, but you can't access that feature from JavaScript. As a consequence, the file you save to disk isn't encrypted unless the encryption is part of the standard platform method for doing things. Never assume that you can perform special platform tasks with JavaScript. Even so, you probably won't even miss these special features, because JavaScript works fine without them.

Seeing How JavaScript Fits into an HTML Document

Now that you know a little more about JavaScript, it's time to see JavaScript in action. The following sections guide you through the process of creating a simple HTML5 document and adding some JavaScript code to it. You don't need to understand the underlying theory of why this application works yet. In addition, you don't need to fully understand the JavaScript language *constructs* (keywords used to build a Java application) yet — just follow along with the simple example to see what JavaScript can do.

This section is a lot more fun when you try the example in a number of browsers. Yes, you can get the gist of what's happening by using a single browser, but it will amaze you to see the application perform the same way no matter which browser you use. To get the most out of the following sections, try the example in two or more of your favorite HTML5-compatible browsers that include JavaScript support.

Starting an HTML5 document

HTML has gone through a lot of changes over the years. In order to identify the various kinds of HTML, the World Wide Web Consortium (W3C) has created a number of specifications that define precisely what an HTML document of a particular type should look like. These standards are publicly available — although no one but a computer scientist can really understand them. You can see the HTML5 standard at `http://www.w3.org/TR/2011/WD-html5-20110525/`.

Make sure you get the full benefit of using this book by downloading the companion source code from `http://www.dummies.com/go/html5programmingwithjavascript`. The companion source code will greatly enhance your experience with the book and make working with JavaScript considerably easier. Make sure you also check the blog entries for this book at `http://blog.johnmuellerbooks.com/categories/263/html5-programming-with-javascript-for-dummies.aspx`. The blog entries answer commonly asked questions, provide additional examples, and help you better use the book content to perform tasks.

For the purposes of this book, you can start any HTML5 document like this (you can access this entire example in the `Test.HTML` file found in the `\Chapter 01\Simple JavaScript Example` folder of the downloadable source code for this book):

```
<!DOCTYPE html>
<html>
   <head>
      <title>JavaScript Example</title>
   </head>
   <body>
      <h1>My First JavaScript Example</h1>
      <p>This is a JavaScript test.</p>
   </body>
</html>
```

Each section of this example performs a specific task. For example, the `<!DOCTYPE html>` declaration tells you that this is an HTML5 document. Other sorts of HTML documents have other declarations. When a browser that understands HTML5 sees this declaration, it treats the rest of the document as an HTML5 document and allows use of HTML5 features.

The `<html>` tag begins and ends the document as a whole. Every HTML document also includes two other tags: a `<head>` tag where you place heading information (such as the page's title), and a `<body>` tag where you place the content you want displayed to the end user. This document includes a `<h1>` tag (first-level heading) and a `<p>` tag (paragraph). Figure 1-1 shows how this document looks in Firefox on a Windows 7 system. (Your screen may look a little different.)

Figure 1-1: A typical view of a simple HTML5 document.

Understanding the alert() function

The first bit of JavaScript code you discover in this book is the `alert()` function. All that this function does is display a message box. You probably see the `alert()` function used on sites you visit several times a day because most developers use it relatively often to display updates and other information. Given the utility of the `alert()` function, it's a good addition to your JavaScript toolbox. The `alert()` function takes a message as input.

Creating the JavaScript examples

This book contains a lot of JavaScript examples. You can type them if you want using any text editor or an editor designed specifically for working with JavaScript such as Komodo Edit (`http://www.activestate.com/komodo-edit`). The important thing is that the editor creates text without any formatting. If the editor adds formatting, then the JavaScript interpreter won't be able to read the file. Each example will include a suggested filename that you should use for your example. Simply save the file to a location you can easily find on your hard drive and then open it using your browser. Opening the file in your browser will cause the JavaScript to run automatically so that you can see how your code works. Chapter 2 provides more information on tools you can use to make your experience working with JavaScript a lot easier.

Using the <script> tag

It's time to try the `alert()` function out. Type the following line of code after the `<p>This is a JavaScript test.</p>` line of code in your initial Web page:

```
<script language="JavaScript">
   alert("Hello World");
</script>
```

The `<script>` tag tells the browser to treat the text that follows as a script, rather than as text for display. The `language` attribute tells the browser that the code is in JavaScript and not some other language. When you display the page in a browser, the user sees a dialog box like the one shown in Figure 1-2.

Figure 1-2: The alert() function displays a simple message box.

To dismiss the message box, the user simply clicks OK. There's nothing fancy about the `alert()` function, but it can convey simple messages, and it's so standard that any browser can display it, even if the browser wouldn't ordinarily work well with newer versions of JavaScript. Use the `alert()` function when you need to tell the user something and don't need to obtain any input in return.

Placing the code in the page heading

Creating a script that runs immediately when you display the page probably works in some cases, but not in others. For example, you may not have anything to say to the user until the user performs some action. In this case, you place the script between the beginning and ending of the `<head>` tag. You also give the script a name so that you can access it at any time. Add this code under the `<title>` tag in the page you created earlier:

```
<script language="JavaScript">
   function SayHello()
   {
      alert("This is the SayHello() function!");
   }
</script>
```

As in the preceding section, you place the script within a `<script>` tag and tell the browser what language you're using to create the script. The `function` keyword tells the browser that this is a particular section of named code, which has a name of `SayHello` in this case. The curly braces (`{}`) tell the browser where the script code begins and ends. In this case, the script consists of a single line of code that contains the `alert()` function.

You could save the page at this point, and it would load just fine, but you can't access the `SayHello()` function. To access the `SayHello()` function, you must provide content that tells the browser to perform the tasks that are contained within the function. To make this happen, add the following lines of code after the `<p>` tag in `<body>` section of the page:

```
<input type="button"
       value="Click Me"
       onclick="SayHello()" />
```

This form of the `<input>` tag creates a button (specified by the `type` attribute). The button has Click Me as a caption as specified by the `value` attribute. When the user clicks the button, the browser performs the task defined by the `SayHello()` function as specified by the `onclick` attribute. Load the page in your browser and dismiss the initial message box. You see the button added to the page, as shown in Figure 1-3.

The `<input>` tag can create a number of controls on a page — buttons are only one such control. You change the kind of control that `<input>` creates through the `type` attribute. Later chapters show more of the `<input>` tag options at your disposal. For now, all you need to know is that `<input>` is a handy tag type to know about.

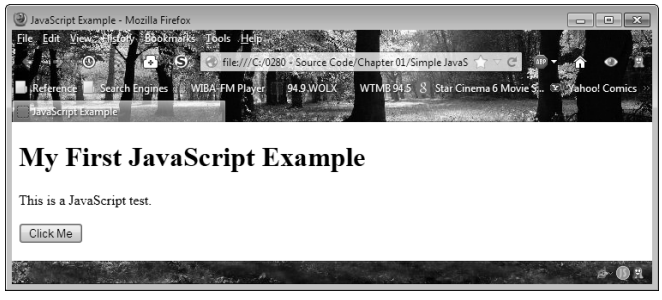

Figure 1-3: The <input> tag lets you add a button to the page.

The advantage of using named code and a button is that you can access the message box as often as needed. Whenever the user clicks Click Me, the browser displays the message box shown in Figure 1-4. Try it now. You must dismiss the dialog box before the browser returns control to the page, but you can display the dialog box as many times as desired.

Figure 1-4: You can display this dialog box as often as desired without reloading the page.

Relying on external files

When you use a particular script regularly, you can do one of two things:

- Use cut and paste techniques to place the script everywhere you need it.
- Place the script in an external file.

The problem with cutting and pasting is that you end up with lots of copies of the same script. If you need to make a change to the script, you have to change every copy you create, which is error prone and time consuming. Using an external file means that you create the script only once and then use it everywhere. The script is easy to change because you change it in only one location.

Begin this part of the example by creating a new file using any means you like (such as a favorite text editor or an application specially designed for working with JavaScript). Name it `External.JS`. JavaScript files normally have a `.JS` file extension. Place this code inside the `External.JS` file:

```
function ExternalSayHello()
{
    alert("This is the ExternalSayHello() function!");
}
```

This code functions exactly like the code that appears in the `<head>` tag of the example page. It displays a message box using the `alert()` function. However, the functions you create in `External.JS` must have unique names. You can't have two functions with the same name in the same page. Notice that this function has a name of `ExternalSayHello` to differentiate it from the `SayHello()` function you created earlier in the chapter.

You have to tell the page where to access this code. To do this, you create a different sort of `<script>` tag entry in the `<head>` tag area of the page. This `<script>` tag looks like this:

```
<script src="External.JS">
</script>
```

The `src` attribute tells the browser to load all of the code found in `External.JS`. You access any function that appears in `External.JS` precisely the same way you would any code that appears in the `<head>` tag. To see how this works, add a new button directly after the first button you created in the preceding section using the following code:

```
<input type="button"
       value="Test External"
       onclick="ExternalSayHello()" />
```

This button works and acts precisely the same as the other button you created. The only difference is that it calls `ExternalSayHello()` instead of `SayHello()` when the user clicks the button. Figure 1-5 shows how the page looks with the additional button on it.

Figure 1-5: The page has two buttons on it now.

Unless you provide additional formatting, the browser simply places the buttons side by side on the page as shown. When the user clicks Test External, the browser displays the message box shown in Figure 1-6. As with the Click Me button, you can display this message box as often as needed.

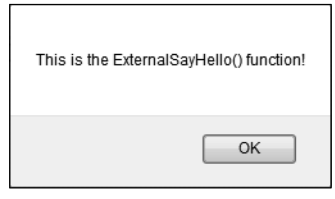

Figure 1-6: Click Test External to see this message box.

Table of Contents

Introduction 1

About This Book 1
What You Don't Have to Read 2
Foolish Assumptions 2
Conventions Used in This Book 3
How This Book Is Organized 3
Part I: Understanding the Basics of JavaScript 3
Part II: Speaking the JavaScript Language 4
Part III: Meeting JavaScript's Control Structures 4
Part IV: Interacting with Users and HTML 4
Part V: Extending JavaScript Further 5
Part VI: The Part of Tens 5
The companion Website 5
Icons Used in This Book 6
Where to Go from Here 6

Part 1: Understanding the Basics of JavaScript 9

Chapter 1: HTML, Say Hello to JavaScript 11

Introducing JavaScript 12
Java and JavaScript aren't long-lost relatives 13
Recognizing the benefits of JavaScript 13
Seeing How JavaScript Fits into an HTML Document 16
Starting an HTML5 document 16
Understanding the alert() function 17
Using the <script> tag 18
Placing the code in the page heading 19
Relying on external files 20

Chapter 2: Assessing Tools of the Trade 23

Researching Browsers and Their Debugging Tools 24
Catching up with Firefox 25
Meeting Google Chrome 29
Checking out native browsers: Internet Explorer or Safari 33
Checking browser and version 34

Discovering Programs to Write JavaScript 36
Using a text editor 37
Using a dedicated editor 38
Hosting Your Site 41
Quickly looking at Web hosting 41
Using JavaScript from your computer 42
Testing your installation 46

Chapter 3: Integrating HTML5 and JavaScript 47

Creating Simple Output 48
Writing to an HTML element 48
Creating direct document output 50
Avoiding potential problems with output 51
Working with JavaScript Statements 52
Understanding the dot syntax 52
Adding multiple statements to a single script 53
Defining basic code blocks 55
Understanding case sensitivity 55
Using white space effectively 55
Using the line continuation character 56
Writing Comments in JavaScript 57
Creating single-line comments 58
Creating multi-line comments 59
Preventing code execution by using comments 60
Alerting Visitors That JavaScript Has Something to Say 60
Using the <noscript> tag 60
Using styles 61

Part II: Speaking the JavaScript Language 63

Chapter 4: Embracing JavaScript Variables 65

Understanding Simple Variables 66
Seeing variables as storage bins 66
Declaring variables 67
Discovering variable types 68
Understanding undefined and null variables 70
Working with Booleans 71
Working with Numbers 72
Understanding the operators 73
Doing simple math 78
Changing number formats 80
Working with Text 83
Concatenating text 83
Changing word and letter formats 83

Working with Arrays 85
Creating an array 85
Accessing array members 86

Chapter 5: Working with Objects 89

Defining Objects 90
Starting with the Object object 90
Understanding properties 92
Understanding methods 92
Understanding events 92
Using and Creating Objects 93
Using JavaScript objects 93
Building custom objects 98
Working with Object Literals 100
Creating the name/value pairs 100
Adding new name/value pairs 102
Naming Variables and Objects Properly 102

Chapter 6: Getting to Know the Standard JavaScript Objects 103

Defining the Basic JavaScript Objects 104
Understanding Object 104
Understanding Number 106
Understanding String 107
Understanding Date 110
Understanding RegExp 114
Introducing the String Object 118
Manipulating and searching strings 118
Working with regular expressions 119
Working with the Date Object 121
Getting today's date right 121
Calculating differences in dates 123

Part III: Meeting JavaScript's Control Structures 125

Chapter 7: Using Functions 127

Using the Built-In Functions 127
Working with the Math Object 130
Building Custom Functions 132
Reducing work using functions 132
Passing parameters to functions 133
Creating functions with a return value 134
Creating functions with optional and required arguments 135
Understanding variable scope 136
Working with Private Properties and Methods 137

Chapter 8: Making Choices in JavaScript 141

Understanding the Reasons for Applications to Make Decisions 142
Deciding When to Use the if Statement 143
Creating a simple if statement 143
Deciding what else to do 144
Nesting if statements 144
Switching Between Choices 146
Creating a basic switch 147
Using the default option 148

Chapter 9: Making the Rounds with Loops 151

Discovering Loops 152
Learning why loops are useful 152
Running from the dangers of loops 153
Creating for Loops 153
Making your first for loop 154
Using the break statement 155
Using the continue statement 157
Creating while Loops 160
Using the basic while loop 161
Using the do...while loop 162
Examining Objects Using for/in 163
Nesting Loops 165

Chapter 10: Performing Error Handling 167

Understanding Errors 168
Defining an error source 168
Defining the error types 170
Catching Errors 171
Using the try...catch block 172
Understanding the Error object 173
Throwing Errors 174

Part IV: Interacting with Users and HTML 177

Chapter 11: Understanding the Document Object Model 179

Introducing the Document Object Model (DOM) 180
Accessing Individual HTML Elements 183
Modifying the output stream 183
Affecting HTML content 186
Changing attributes 186
Considering Events 188
Working with Styles 191

Working with Nodes 192
Creating new elements 192
Removing existing elements 194

Chapter 12: Handling Events 195

Introducing Events 196
Performing basic event tasks 196
Working with attributes 200
Clicking to Create an Event 202
Understanding the objects at your disposal 202
Passing parameters to the event handler function 208
Pressing a Key 210
Creating Custom Events 212
Working with the CustomEvent object 212
Creating the custom event code 212

Chapter 13: Connecting with Style: JavaScript and CSS 215

Changing HTML Elements 216
Working with HTML tags 216
Working with heading styles 218
Working with IDs 219
Building Dynamic HTML Elements 219
Animating and Positioning HTML Elements 221
Creating JavaScript-Based Menus 221
Designing the HTML 222
Defining the styles 223
Creating the JavaScript functions 224

Chapter 14: Enhancing HTML5 Forms with JavaScript 227

Revisiting the HTML5 Form Features 228
Working with the new elements 228
Working with the new controls 230
Understanding the new attributes 233
Accessing Form Elements 235
Validating Forms 237

Chapter 15: Interacting with Windows 241

Working with Windows 242
Determining window characteristics 242
Changing window attributes 243
Accessing the Screen 244
Finding a Location 245
Determining the current location 245
Changing the location 246
Searching History 247

Working with Navigator 247
Creating a Pop-Up 249
Controlling line breaks 249
Creating a modal dialog box 250
Using Cookies 253
Setting a cookie 254
Getting a cookie 254
Checking a cookie 256

Part V: Extending JavaScript Further 257

Chapter 16: Working with XML in JavaScript 259

Introducing XML 260
Gaining an overview of XML 260
Understanding elements 261
Understanding attributes 263
Working through a basic file 264
Displaying XML Using XSLT 265
Validating XML 267
Understanding the concept of well-formed 268
Discovering XSD 268
Loading XML with JavaScript 269

Chapter 17: Cleaning Up the Web with AJAX 271

Introducing AJAX 271
Learning the benefits of AJAX 272
Understanding how AJAX works 272
Deciphering the XMLHttpRequest object 273
Performing AJAX Tasks Using JavaScript 276
Making AJAX Easier with jQuery 278

Chapter 18: Making JavaScript Easier with jQuery 279

Getting to Know the jQuery Library 280
Loading jQuery from Google CDN 282
Doing Things Easier with jQuery 284
Gathering elements with selectors 284
Working with programmatically generated elements 286
Revisiting arrays with jQuery 288
Interrogating an array 289
Handling Events with jQuery 290
Understanding the event functionality 290
Working with events 296

Chapter 19: Using jQuery to Handle AJAX 299

Understanding the AJAX Object in jQuery 300
Considering the global AJAX event handlers 300
Working with the helper functions 301
Understanding the low-level interface 302
Using shorthand methods 303
Discovering Server-Side Programming 304
Adding PHP support to your Apache server 304
Testing your PHP configuration 306
Creating the PHP script 306
Retrieving Results from AJAX Calls 308
Working with standard output 308
Investigating the benefits of JSON 310

Chapter 20: Animating the Web 313

Getting to Know jQuery UI 314
Considering interactions 314
Understanding the widgets 322
Defining the effects 333
Canvasing Your Web Page 340
Understanding what the canvas does 340
Creating a simple Google API application 341

Part VI: The Part of Tens 345

Chapter 21: Ten Incredible HTML5 and JavaScript Examples 347

Creating an XML-Based Menu 348
Creating an XML-Based Tabbed Interface 352
Displaying a Slideshow 355
Employing Custom Spinners 358
Working with Timepickers 361
Creating a Standardized Look with CSS 362
Displaying a Category and Detail Data View 363
Selecting Data Using a Calendar 367
Developing a User-Configurable Interface 370
Devising a Simple Survey Form 371

Chapter 22: Ten Thoughts About the Future of Web Development . . . 375

Automating More Tasks 375
Developing Applications That Run Anywhere 376
Using Standards for Every Application 376
Creating a Desktop Environment with a Browser 377

Using a Centralized Data Store 377
Creating Mobile-friendly Applications 378
Developing Accessible Applications 378
Building New Application Types 379
Thinking More About Users 379
Expecting Unexpected Connections 380

Index *381*