Brad Dayley

Sams Teach Yourself

jQuery and JavaScript in 24 Hours

SAMS

Brad Dayley

Sams Teach Yourself

jQuery and JavaScript

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself jQuery and JavaScript in 24 Hours

Copyright © 2014 by Pearson Education, Inc.

ISBN-13: 978-0-672-33734-5

ISBN-10: 0-672-33734-7

Library of Congress Control Number: 2013954604

Printed in the United States of America

2 17

Trademarks

Warning and Disclaimer

Special Sales

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Barbara Hacha

Indexer
Erika Millen

Proofreader
Anne Goebel

Technical Editor
Russell Kloepfer

Publishing Coordinator
Vanessa Evans

Book Designer
Gary Adair

Cover Designer
Mark Shirar

Compositor
Jake McFarland

For D!

—A & F

Contents at a Glance

Table of Contents

About the Author

Brad Dayley is a senior software engineer with more than 20 years of experience developing enterprise applications. He has used HTML/CSS, JavaScript, and jQuery extensively to develop a wide array of web pages, ranging from enterprise application interfaces to sophisticated, rich Internet applications, to smart interfaces for mobile web services. He is the author of *Python Phrasebook* and *jQuery and JavaScript Phrasebook*.

Acknowledgments

I'd like to take this opportunity to thank all those who made this title possible. First, thanks to my wonderful wife and boys for giving me the inspiration and support I need. I'd never make it far without you.

Thanks to Mark Taber for getting this title rolling in the right direction, Russell Kloepfer, for keeping me honest with his technical review, Barbara Hacha, for turning the technical ramblings of my brain into a fine text, and Tonya Simpson, for managing everything on the production end and making sure the book is the finest quality.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email:	feedback@samspublishing.com
Mail:	Sams Publishing ATTN: Reader Feedback 800 East 96th Street Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

With billions of people using the Internet today, there is a rapidly growing trend to replace traditional websites, where pages link to other pages with a single page, with applications that have richly interactive elements. The main reason for this is that users have become less patient with clicking, waiting, and then having to navigate back and forth between web pages. Instead, they want websites to behave more like the applications they are used to on their computers and mobile devices.

In fact, in just the next 24 hours, millions of new web pages will be added to the Internet. The majority of these pages will be written in HTML, with CSS to style elements and with JavaScript to provide interaction between the user and back-end services.

As you complete the 24 one-hour lessons in this book, you will gain a practical understanding of how to incorporate JavaScript with the powerful jQuery library to provide rich user interactions in your web pages. You will gain the valuable skills of adding dynamic code that allows web pages to instantly react to mouse clicks and finger swipes, interact with back-end services to store and retrieve data from the web server, and create robust Internet applications.

Each hour in the book provides fundamentals that are necessary to create professional web applications. The book includes some basics on using HTML and CSS to get you started, even if you've never used them before. You are provided with code examples that you can implement and expand as your understanding increases. In fact, in just the first lesson in the book, you create a dynamic web page using jQuery and JavaScript.

So pull up a chair, sit back, and enjoy the ride of programming rich Internet applications with jQuery and JavaScript.

Beyond jQuery and JavaScript

This book covers more than jQuery and JavaScript because you need to know more than the language structure to create truly useful web applications. The goal of this book is to give you the fundamental skills needed to create fully functional and interactive web applications in just 24 short, easy lessons. This book covers the following key skills and technologies:

- HTML is the most current recommendation for web page creation. Every example in this book is validated HTML5, the most recent recommended version.
- CSS is the standard method for formatting web elements. You not only learn how to write CSS and CSS3, but also how to dynamically modify it on the fly using jQuery and JavaScript.
- JavaScript is the best method to provide interactions in web pages without the need to load a new page from the server. This is the standard language on which most decent web applications are built.
- jQuery, jQueryUI, and jQueryMobile are some of the most popular and robust libraries for JavaScript. jQuery provides very quick access to web page elements and a robust set of features for web application interaction. jQuery provides additional UI and mobile libraries that provide rich UI components for traditional web applications as well as mobile web applications.
- AJAX is the standard method that web applications use to interact with web servers and other services. The book includes several examples of using AJAX to interact with web servers, Google, Facebook, and other popular web services.

Code Examples

Most of the examples in the book provide the following elements:

- **HTML code**—Code necessary to provide the web page framework in the browser.
- **CSS code**—Code necessary to style the web page elements correctly.
- **JavaScript code**—This includes both the jQuery and JavaScript code that provide interactions among the user, web page elements, and web services.
- **Figures**—Most of the examples include one or more figures that illustrate the behavior of the code in the browser.

The examples in the book are basic to make it easier for you to learn and implement. Many of them can be expanded and used in your own web pages. In fact, some of the exercises at the end of each hour have you expand on the examples.

All the examples in the book have been tested for compatibility with the latest version of the major web browsers, including Google's Chrome, Microsoft's Internet Explorer, and Mozilla's Firefox.

Special Elements

As you complete each lesson, margin notes help you immediately apply what you just learned to your own web pages.

Whenever a new term is used, it is clearly explained. No flipping back and forth to a glossary!

TIP

Tips and tricks to save you precious time are set aside in Tips so that you can spot them quickly.

NOTE

Notes highlight interesting information you should be sure not to miss.

CAUTION

When there's something you need to watch out for, you'll be warned about it in a Caution.

Q&A, Quizzes, and Exercises

Every hour ends with a short question-and-answer session that addresses the kind of "dumb questions" everyone wants to ask. A brief but complete quiz lets you test yourself to be sure you understand everything presented in the hour. Finally, one or two optional exercises give you a chance to practice your new skills before you move on.

HOUR 1
Intro to Dynamic Web Programming

What You'll Learn in This Hour:

- ▶ Getting ready for creating dynamic web pages
- ▶ Creating a jQuery- and a JavaScript-friendly development environment
- ▶ Adding JavaScript and jQuery to web pages
- ▶ Constructing web pages to support jQuery and JavaScript
- ▶ Creating your first dynamic web pages with jQuery and JavaScript

JavaScript and its amped up counterpart jQuery have completely changed the game when it comes to creating rich interactive web pages and web-based applications. JavaScript has long been a critical component for creating dynamic web pages. Now, with the advancements in the jQuery, jQuery UI, and jQuery Mobile libraries, web development has changed forever.

This hour quickly takes you through the world of jQuery and JavaScript development. The best place to start is to ensure that you understand the dynamic web development playground that you will be playing in. To be effective in JavaScript and jQuery, you need a fairly decent understanding of web server and web browser interaction, as well as HTML and CSS.

This hour includes several sections that briefly give a high-level overview of web server and browser interactions and the technologies that are involved. The rest of this hour is dedicated to setting up and configuring a jQuery and JavaScript friendly development environment. You will end with writing your very first web pages that include JavaScript and jQuery code.

Understanding the Web Server/Browser Paradigm

JavaScript and jQuery interact with every major component involved in communication between the web server and the browser. To help you understand that interaction better, this section provides a high-level overview of the concepts and technologies involved in web server/browser communication. This is not intended to be comprehensive by any means; it's a high-level overview that enables you to put things into the correct context as they are discussed later in the book.

Looking at Web Server to Browser Communication Terms

The World Wide Web's basic concept should be very familiar to you: An address is typed into or clicked in a web browser, and information is loaded in a form ready to be used. The browser sends a request, the server sends a response, and the browser displays it to the user.

Although the concept is simple, several steps must take place. The following sections define the components involved, their interactions with each other, and how JavaScript and jQuery are involved.

Web Server

The web server is the most critical component of the web. Without it, no data would be available at all. The web server responds to requests from browsers with data that the browsers then display. A lot of things happen on the web server, though. For example, the web server and its components check the format and validity of requests. They may also check for security to verify that the request is from an allowed user. Then, to build the response, the server may interact with several components and even other remote servers to obtain the data necessary.

Browser

The next most important component is the browser. The browser sends requests to the web server and then displays the results for the user. The browser also has a lot of things happening under the hood. The browser has to parse the response from the server and then determine how to represent that to the user.

Although several browsers are available, the three most popular are Firefox, Internet Explorer, and Chrome. For the most part, each browser behaves the same when displaying web pages; however, occasionally some differences exist, and you will need to carefully test your JavaScript and jQuery scripts in each of the browsers that you want to support.

JavaScript and jQuery can be very involved in the interactions that occur between the browser receiving the response and the final output rendered for the user. These scripts can change the format, content, look, and behavior of the data returned from the server. The following sections describe important pieces provided by the browser.

DOM

The browser renders a web page by creating a Document Object Model, or DOM. The DOM is a tree structure with the HTML document as the root object. The root can have several children, and those children can have several children. For example, a web page that contains a list would have a root object, with a child list object that contained several child list element objects. The following shows an example of a simple DOM tree for a web page containing a single heading and a list of three cities:

```
document
  + html
    + body
      + h1
        + text = "City List"
      + ul
        + li
          + text = "New York, US"
        + li
          + text = "Paris, FR"
        + li
          + text = "London, EN"
```

The browser knows how to display each node in the DOM and renders the web page by reading each node and drawing the appropriate pixels in the browser window. As you learn later, JavaScript and jQuery enable you to interact directly with the DOM, reading each of the objects, changing those objects, and even removing and adding objects.

Browser Events

The browser tracks several events that are critical to jQuery and JavaScript programs—for example, when a page is loaded, when you navigate away from a page, when the keyboard is pressed, mouse movements, and clicks. These events are available to JavaScript, allowing you to execute functionality based on which events occur and where they occur.

Browser Window

The browser also provides limited access to the browser window itself. This allows you to use JavaScript to determine the display size of the browser window and other important information that you can use to determine what your scripts will do.

URL

The browser is able to access files on the web server using a Uniform Resource Locator, or URL. A URL is a fully unique address to access data on the web server, which links the URL to a specific file or resource. The web server knows how to parse the URL to determine which file/resources to use to build the response for the browser. In some instances, you might need to use JavaScript to parse and build URLs, especially when dynamically linking to other web pages.

HTML/HTML5

Hypertext Markup Language, or HTML, provides the basic building blocks of a web page. HTML defines a set of elements representing content that is placed on the web page. Each element is enclosed in a pair of tags denoted by the following syntax:

```
<tag>content</tag>
```

For example:

```
<p>This is an HTML paragraph.</p>.
```

The web browser knows how to render the content of each of the tags in the appropriate manner. For example, the tag `<p>` is used to denote a paragraph. The actual text that is displayed on the screen is the text between the `<p>` start tag and the `</p>` end tag.

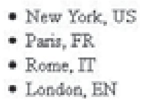
The format, look, and feel of a web page is determined by placement and type of tags that are included in the HTML file. The browser reads the tags and then renders the content to the screen as defined.

HTML5 is the next generation of the HTML language that incorporates more media elements, such as audio and video. It also provides a rich selection of vector graphic tags that allow you to draw sharp, crisp images directly onto the web page using JavaScript.

Listing 1.1 shows an example of the HTML used to build a simple web page with a list of cities. The HTML is rendered by the browser into the output shown in Figure 1.1.

LISTING 1.1 **A Simple HTML Document That Illustrates the HTML Code Necessary to Render a List in a Browser**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cities</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```



- New York, US
- Paris, FR
- Rome, IT
- London, EN

FIGURE 1.1
List of cities rendered in a browser using the code from Listing 1.1.

CSS

One of the challenges with web pages is getting them to look sharp and professional. The generic look and feel that browsers provide by default is functional; however, it is a far cry from the sleek and sexy eye-candy that users of today's Internet have come to expect.

Cascading Style Sheets, or CSS, provide a way to easily define how the browser renders HTML elements. CSS can be used to define the layout as well as the look and feel of individual elements on a web page. To illustrate this, I've added some CSS code to our example from Listing 1.1. Listing 1.2 uses CSS to modify several attributes in lines 07 to 13. These attributes alter the text alignment, font style, and change the list bullet from a dot to an airplane image. Notice how the CSS style changes how the list is rendered in Figure 1.2.

LISTING 1.2 HTML with Some CSS Code in `<STYLE>` Element to Alter the Appearance of the List

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
05      <style>
06       li {
07         text-align: center;
08         font-family: "Times New Roman", Times, serif;
09         font-size: 30px;
10         font-style: italic;
11         font-weight: bold;
12         list-style-type: none;
13         list-style-image: url('images/air.png');
14       }
15     </style>
16   </head>
17   <body>
18     <ul>
19       <li>New York, US</li>
20       <li>Paris, FR</li>
21       <li>Rome, IT</li>
22       <li>London, EN</li>
```

```
    </ul>
  </body>
</html>
```

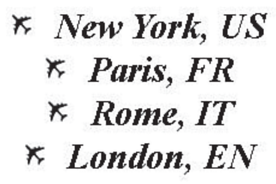

New York, US
Paris, FR
Rome, IT
London, EN

FIGURE 1.2
The CSS code dramatically changes the look of the list in the browser.

HTTP/HTTPS Protocols

Hypertext Transfer Protocol (HTTP) defines communication between the browser and the web server. It defines what types of requests can be made, as well as the format of those requests and the HTTP response.

Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) adds an additional security layer, SSL/TLS, to ensure secure connections. When a web browser connects to a web server via HTTPS, a certificate is provided to the browser. The user is then able to determine whether to accept the certificate. Without the certificate, the web server will not respond to the user's requests, thus ensuring that the request is coming from a secured source.

The following sections discuss a little bit about HTTP headers and the two most common types of HTTP requests, GET and PUT.

HTTP Headers

HTTP headers allow the browser to define the behavior and format of requests made to the server and the response back to the web browser. HTTP headers are sent as part of an HTTP request and response. You can send HTTP requests to web servers from JavaScript, so you need to know at least a little bit about the headers required.

The web server reads the request headers and uses them to determine how to build a response for the browser. As part of the response, the web server includes response headers that tell the browser how to process the data in the response. The browser reads the headers first and uses the header values when handling the response and rendering the page.

Following are a few of the more common ones:

- `ACCEPT`—Defines content types that are acceptable in the response.

- `AUTHORIZATION`—Specifies authentication credentials used to authenticate the requesting user.
- `COOKIE`—Cookie value that was previously set in the browser by a server request. Cookies are key/value pairs that are stored on the client. They can be set via server requests or JavaScript code and are sent back to the server as part of HTTP requests from the browser.
- `SET-COOKIE`—Cookie value from the server that the browser should store if cookies are enabled.
- `CONTENT-TYPE`—Type of content contained in the response from the web server. For example, this field may be "`text/plain`" for text or "`image/png`" for a .png graphic.
- `CONTENT-LENGTH`—Amount of data that is included in the body of the request or response.

Many more headers are used in HTTP requests and responses, but the preceding list should give you a good idea of how they are used.

GET Request

The most common type of HTTP request is the GET request. The GET request is generally used to retrieve information from the web server—for example, to load a web page or retrieve images to display on a web page. The file to retrieve is specified in the URL that is typed in the browser, for example:

```
http://www.dayleycreations.com/tutorials.html
```

A GET request is composed entirely of headers with no body data. However, data can be passed to the server in a GET request using a query string. A query string is sent to the web server as part of the URL. The query string is formatted by specifying a ? character after the URL and then including a series of one or more key/value pairs separated by & characters using the following syntax:

```
URL?key=value&key=value&key=value...
```

For example, the following URL includes a query string that specifies a parameter `gallery` with a value of `01` that are sent to the server:

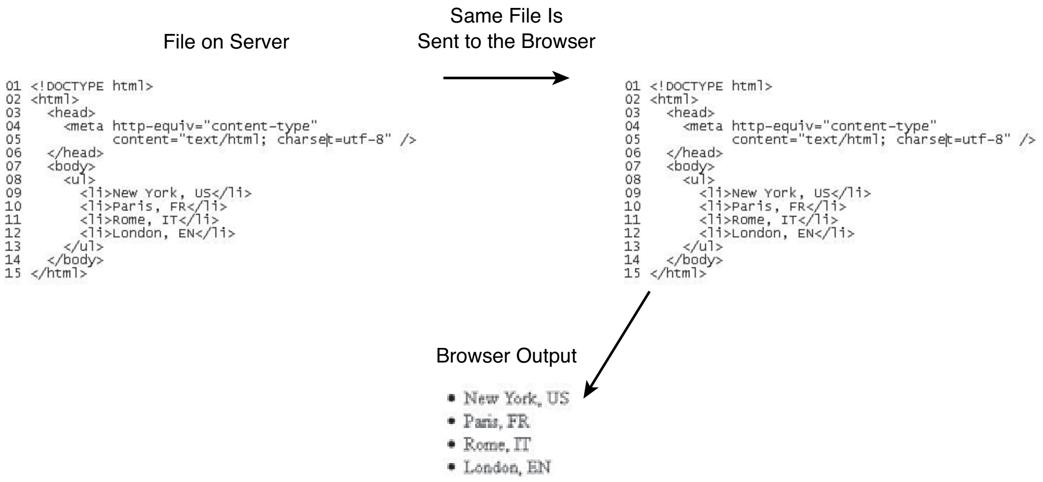
```
http://www.dayleycreations.com/gallery.html?gallery=01
```

POST Request

A POST request is different from a GET request in that there is no query string. Instead, any data that needs to be sent to the web server is encoded into the body of the request. POST requests are generally used for requests that change the state of data on the web server. For example, a web form that adds a new user would send the information that was typed into the form to the server as part of the body of a POST.

Web Server and Client-Side Scripting

Originally, web pages were static, meaning that the file that was rendered by the browser was the exact file that was stored on the server, as shown in Figure 1.3. The problem is that when you try to build a modern website with user interactions, rich elements, and large data, the number of web pages needed to support the different static web pages is increased dramatically.



```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type"
          content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type"
          content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

- New York, US
- Paris, FR
- Rome, IT
- London, EN

FIGURE 1.3
With static pages, the same page that is located on the web server is sent to the browser and rendered directly.

Rather than creating a web server full of static HTML files, it is better to use scripts that use data from the web server and dynamically build the HTML that is rendered in the browser.

Those scripts can either run on the server or in the client browser. The following sections discuss each of those methods. Most modern websites use a combination of server- and client-side scripting.

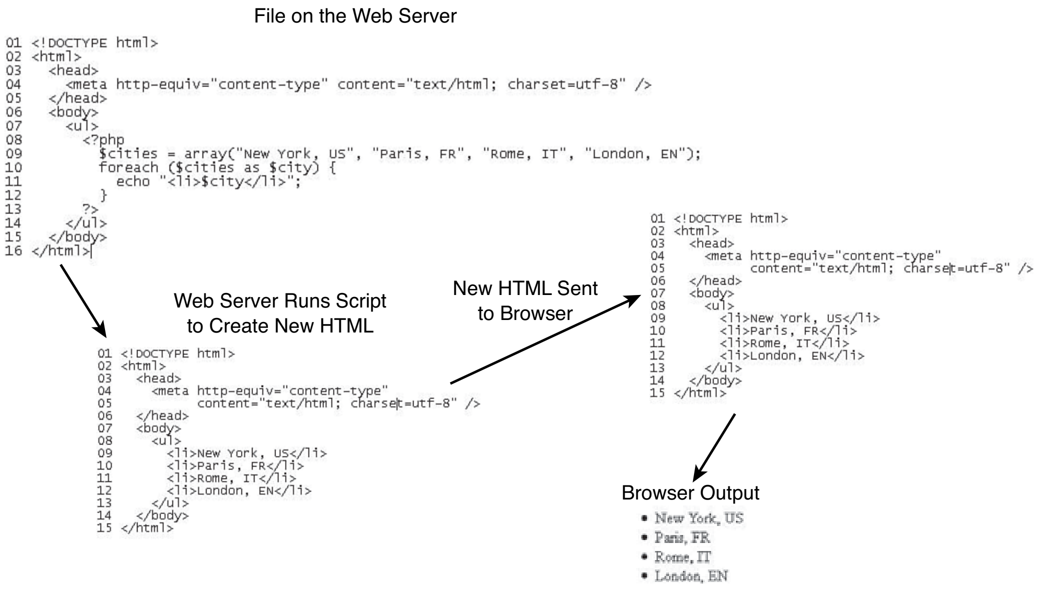
Server-Side Scripting

Server-side scripting is the process of formatting server data into an HTML response before it is sent back to the browser. The main advantages of server-side scripting are that data processing is done completely on the server side and the raw data is never transferred across the Internet; also, problems and data fix-ups can be done locally within the server processing. The disadvantage is that it requires more processing on the server side, which can reduce the scalability of some applications. Listing 1.3 shows a simple server-side PHP script that dynamically adds the list of cities to an HTML document before sending it to the browser.

Figure 1.4 shows an example of a simple PHP server-side script. Notice that the file located on the server is different from the one sent to the browser, but the same one sent to the browser is what is rendered.

LISTING 1.3 A PHP Script That Is Run at the Server Populates the City List Items

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <?php
        $cities = array("New York, US", "Paris, FR", "Rome, IT", "London, EN");
        foreach ($cities as $city) {
          echo "<li>$city</li>";
        }
      ?>
    </ul>
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <?php
        $cities = array("New York, US", "Paris, FR", "Rome, IT", "London, EN");
        foreach ($cities as $city) {
          echo "<li>$city</li>";
        }
      ?>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type"
          content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type"
          content="text/html; charset=utf-8" />
  </head>
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

```
• New York, US
• Paris, FR
• Rome, IT
• London, EN
```

FIGURE 1.4
The PHP script is executed on the web server, and so the HTML document sent to the browser is different from what is actually contained on the server.

Client-Side Scripting

Client-side scripting is the process of sending code along with the web page. That code gets executed either during the loading of the web page or after the web page has been loaded.

There are a couple of great advantages of client-side scripting. One is that data processing is done on the client side, which makes it easier to scale applications with large numbers of users. Another is that browser events can often be handled locally without the need to send requests to the server. This enables you to make interfaces respond much more quickly to user interaction.

JavaScript and jQuery are by far the most common form of client-side scripting. Throughout this book, you learn why that is the case.

What Is JavaScript?

JavaScript is a programming language much like any other. What separates JavaScript the most from other programming languages is that the browser has a built-in interpreter that can parse and execute the language. That means that you can write complex applications that have direct access to the browser and the DOM.

Access to the DOM means that you can add, modify, or remove elements from a web page without reloading it. Access to the browser gives you access to events such as mouse movements and clicks. This is what gives JavaScript the capability to provide functionality such as dynamic lists and drag and drop.

What Is jQuery?

jQuery is a library that is built on JavaScript. The underlying code is JavaScript; however, jQuery simplifies a lot of the JavaScript code into simple-to-use functionality. The two main advantages to using jQuery are selectors and built-in functions.

Selectors provide quick access to specific elements on the web page, such as a list or table. They also provide access to groups of elements, such as all paragraphs or all paragraphs of a certain class. This allows you to quickly and easily access specific DOM elements.

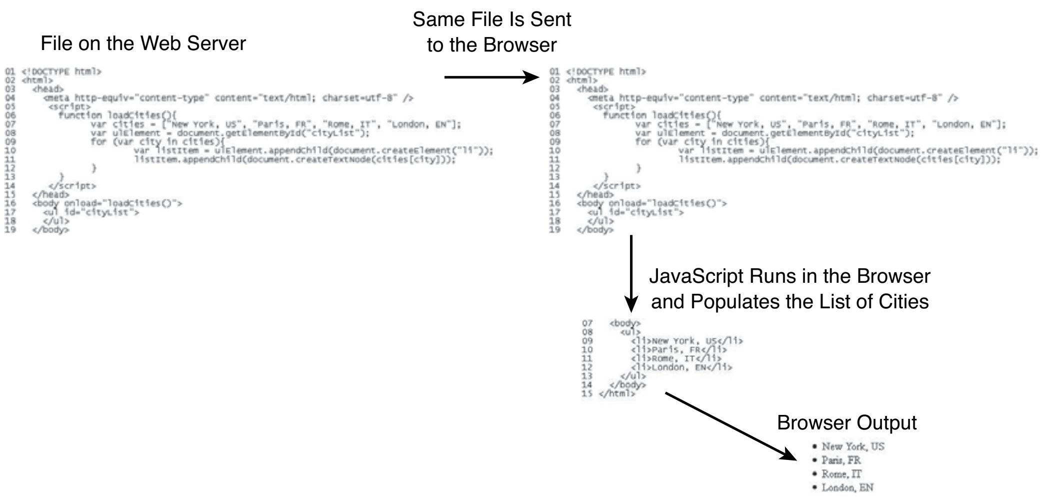
jQuery also provides a rich set of built in functionality that makes it easy to do a lot more with a lot less code. For example, tasks such as hiding an element on the screen or animating the resize of an element take just one line of code.

Client-Side Scripting Example

Listing 1.4 shows an example of a simple JavaScript client-side script. Figure 1.5 diagrams the flow of data between the web server and the browser. Notice that this time the file located on the server is the same one sent to the browser, but the JavaScript changes the HTML that is loaded in the browser.

LISTING 1.4 A Simple JavaScript Client-Side Script That Is Run in the Browser to Populate the City List Items

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
     <script>
       function loadCities(){
             var cities = ["New York, US", "Paris, FR", "Rome, IT", "London,
             ➥EN"];
             var ulElement = document.getElementById("cityList");
             for (var city in cities){
                  var listItem = ulElement.appendChild(document.
                  ➥createElement("li"));
                  listItem.appendChild(document.createTextNode(cities[city]));
             }
       }
     </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
</html>
```


File on the Web Server
Same File Is Sent to the Browser
JavaScript Runs in the Browser and Populates the List of Cities
Browser Output

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
     <script>
       function loadCities(){
             var cities = ["New York, US", "Paris, FR", "Rome, IT", "London, EN"];
             var ulElement = document.getElementById("cityList");
             for (var city in cities){
                  var listItem = ulElement.appendChild(document.createElement("li"));
                  listItem.appendChild(document.createTextNode(cities[city]));
             }
       }
     </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
     <script>
       function loadCities(){
             var cities = ["New York, US", "Paris, FR", "Rome, IT", "London, EN"];
             var ulElement = document.getElementById("cityList");
             for (var city in cities){
                  var listItem = ulElement.appendChild(document.createElement("li"));
                  listItem.appendChild(document.createTextNode(cities[city]));
             }
       }
     </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
```

```
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

```
• New York, US
• Paris, FR
• Rome, IT
• London, EN
```

FIGURE 1.5
The JavaScript is executed in the browser, and so the HTML document rendered by the browser is different from the one that was originally sent.

AJAX

Asynchronous JavaScript and XML, or AJAX, is the process of using JavaScript to continue to communicate with the web server after the web page has been loaded. AJAX reduces the need to reload the web page or load other web pages as the user interacts. This reduces the amount of data that needs to be sent with the initial web server response and also allows web pages to be more interactive.

For a simple example of AJAX, I’ve constructed two scripts—Listing 1.5 and Listing 1.6. Listing 1.5 is an HTML document with JavaScript that runs on the client after the page is loaded. The JavaScript makes an AJAX request back to the server to retrieve the list of cities via a server-side PHP script, shown in Listing 1.6. The list of cities returned is then used to populate the HTML list element with items.

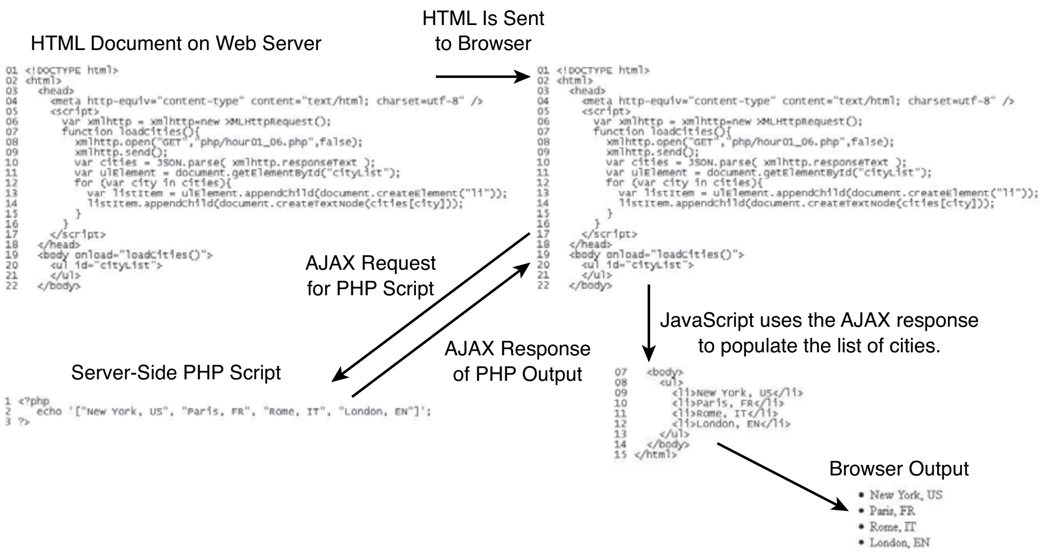
LISTING 1.5 A Simple JavaScript Client-Side Script Executes an AJAX Request to the Server to Retrieve a List of Cities to Use When Building the HTML List Element

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <script>
      var xmlhttp = new XMLHttpRequest();
      function loadCities(){
        xmlhttp.open("GET","php/hour01_06.php",false);
        xmlhttp.send();
        var cities = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("cityList");
        for (var city in cities){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(cities[city]));
        }
      }
    </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
</html>
```

LISTING 1.6 A Server-Side PHP Script That Returns a Simple JSON String That Can Be Parsed and Used by the JavaScript in Listing 1.5

```
<?php
  echo '["New York, US", "Paris, FR", "Rome, IT", "London, EN"]';
?>
```

Figure 1.6 illustrates the flow of communication that happens during the AJAX request/response. Notice that a second request is made to the server to retrieve the list of cities.


HTML Document on Web Server
HTML Is Sent to Browser
AJAX Request for PHP Script
AJAX Response of PHP Output
Server-Side PHP Script
JavaScript uses the AJAX response to populate the list of cities.
Browser Output

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <script>
      var xmlhttp = xmlhttp=new XMLHttpRequest();
      function loadCities(){
        xmlhttp.open("GET","php/hour01_06.php",false);
        xmlhttp.send();
        var cities = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("cityList");
        for (var city in cities){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(cities[city]));
        }
      }
    </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <script>
      var xmlhttp = xmlhttp=new XMLHttpRequest();
      function loadCities(){
        xmlhttp.open("GET","php/hour01_06.php",false);
        xmlhttp.send();
        var cities = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("cityList");
        for (var city in cities){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(cities[city]));
        }
      }
    </script>
  </head>
  <body onload="loadCities()">
    <ul id="cityList">
    </ul>
  </body>
```

```
<?php
    echo '["New York, US", "Paris, FR", "Rome, IT", "London, EN"]';
?>
```

```
  <body>
    <ul>
      <li>New York, US</li>
      <li>Paris, FR</li>
      <li>Rome, IT</li>
      <li>London, EN</li>
    </ul>
  </body>
</html>
```

- New York, US
- Paris, FR
- Rome, IT
- London, EN

FIGURE 1.6
Using an AJAX request, JavaScript can send an additional request to the server to retrieve additional information that can be used to populate the web page.

Preparing to Write jQuery and JavaScript

With the brief introduction to dynamic web programming out of the way, it is time to cut to the chase and get your development environment ready to write jQuery and JavaScript.

The development environment can make all the difference when you are writing jQuery and JavaScript projects. The development environment should have these following characteristics:

- **Easy to Use IDE**—The IDE provides text editors that allow you to modify your code in the simplest manner possible. Choose an IDE that you feel comfortable with and that is extensible to support HTML, CSS, JavaScript, and jQuery.
- **Development Web Server**—You should never develop directly on a live web server (although most of us have done it at one point or another). A web server is required to test out scripts and interactions.

- **Development Web Browser(s)**—Again, you should initially develop to the browser that you are most comfortable with or will be the most commonly used. You will need to enable debugging tools on the browser to help you find and fix issues with your scripts.
- **Well-Structured Project**—Structure your project, directories, and filenames for growth. It is a difficult process to restructure a web project with a large number of files and directories. Too many files or confusing filenames can make a project cumbersome and difficult to manage.

Setting Up a Web Development Environment

Setting up a web development environment requires three steps. First, install an IDE that will provide the tools to create and edit code. Second, add and enable JavaScript debugging tools in your web browser(s). Third, set up a development web server that you can test your scripts from. The following sections take you through each of those tasks.

Installing a Web Development IDE

The IDE is the most important aspect when developing with JavaScript. An IDE integrates the various tasks required to write web applications into a single interface. In reality, you could use any text editor to write HTML, CSS, JavaScript, and jQuery code. However, you will find it much more productive and easy to use a good IDE.

Several IDEs are available. Some are open source; others cost a lot. Pick the one that best fits your needs. You should have an IDE with code completion and error checking, because those two features save the most time.

Possible IDEs include Dreamweaver, Visual Studio, and several others. However, Eclipse features numerous plug-ins that provide extensibility. For jQuery and JavaScript, consider using Aptana Studio because it is simple to set up and get going with, and it is supported on Mac, Windows, and Linux.

Installing Aptana Studio

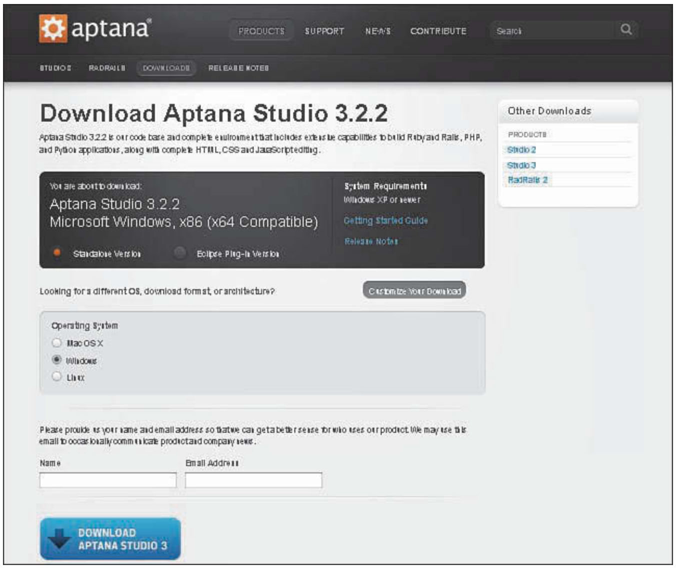
For the purposes of this book, you step through the process of installing and configuring Aptana, although all the editing and debugging concepts apply equally to whatever IDE you are working with.

NOTE

You will need to have a Java JRE or JDK installed to be able to install Aptana Studio.

You can download Aptana Studio from the following location:
www.aptana.com/products/studio3/download

You should select Standalone Version and specify which operating system that you are installing it on, as shown in Figure 1.7. After the install is downloaded, execute it and follow the prompts; they are very straightforward.



Download Aptana Studio 3.2.2

Aptana Studio 3.2.2 is our code base and complete environment that includes extensive capabilities to build Ruby and Rails, PHP, and Python applications, along with complete HTML, CSS and JavaScript editing.

You are about to download:
Aptana Studio 3.2.2
Microsoft Windows, x86 (x64 Compatible)

[✓ Standalone Version] Eclipse Plug-in Version

System Requirements
Windows XP or newer

Looking for a different OS, download format, or architecture?

Operating System
Mac OS X
[✓ Windows]
Linux

Please provide us your name and email address so that we can get a better sense for who uses our product. We may use this email to occasionally communicate product and company news.

Name
Email Address

FIGURE 1.7
Aptana Studio download page.

After Aptana Studio is installed, launch it. When you launch it for the first time, you need to specify a location for your workspace. The workspace is where your projects and files will be stored, so pick a location that you can easily manage.

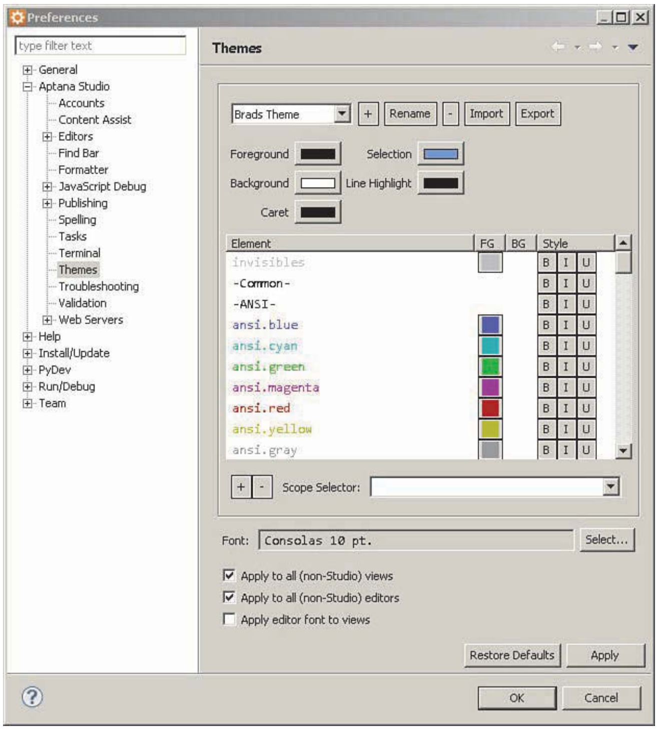
Configuring Aptana Studio

Now that Aptana Studio is installed, there are just a few more steps to configure it. Do the following:

1. Select Commands, Bundle Development, Install Bundle from the main menu. Then select jQuery from the bundles list and click the OK button.
2. Select Windows, Preferences from the main menu to load the Preferences dialog shown in Figure 1.8. From the Preferences dialog select Aptana Studio, Themes, also shown in Figure 1.8, and then select a theme from the drop-down list. The theme will set window, menu, selection, and code element colors and fonts. Choose a theme that works well for you. You can also customize the theme for this page.

NOTE

You can also import and export themes from this themes preferences dialog. On the book website is a file named AptanaTheme.tmTheme, which is the theme used in writing this book. You can import that theme here.



Font: Consolas 10 pt.

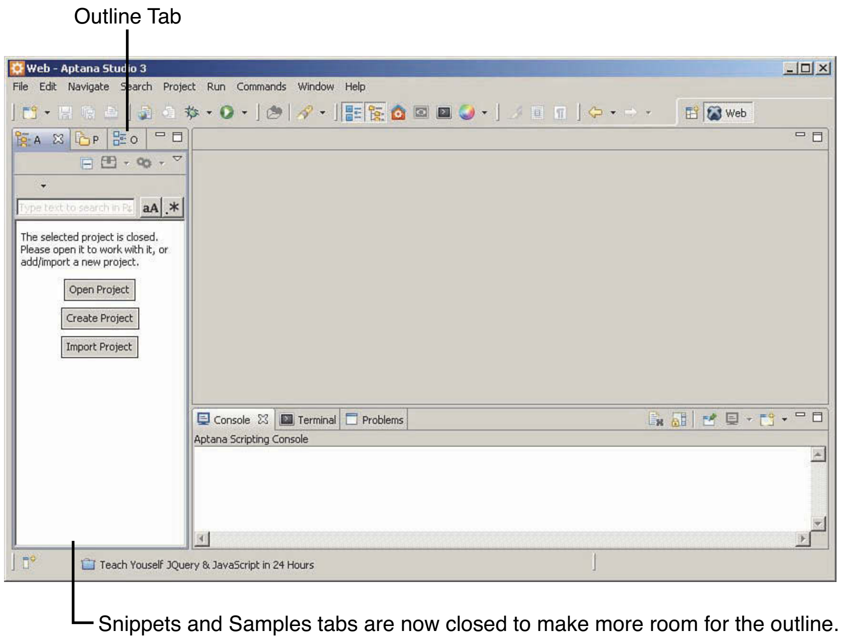
☑ Apply to all (non-Studio) views
☑ Apply to all (non-Studio) editors
☐ Apply editor font to views

FIGURE 1.8
Aptana Studio Preferences dialog.

3. Select and drag the Outline tab in the bottom left and merge it next to the App Explorer tab. This gives you a better view of the outline in your scripts and allows you to close the snippets and sample tabs in the bottom that you do not need. The final result should look something like Figure 1.9.

4. Play around with the preferences and menus to familiarize yourself with the interface.

Aptana Studio is now set up and ready for you to begin creating projects.


Outline Tab
Web - Aptana Studio 3
File Edit Navigate Search Project Run Commands Window Help
Web
Console
Terminal
Problems
Aptana Scripting Console
Open Project
Create Project
Import Project
Teach Youself JQuery & JavaScript in 24 Hours
Snippets and Samples tabs are now closed to make more room for the outline.

The selected project is closed. Please open it to work with it, or add/import a new project.

FIGURE 1.9
Aptana Studio with Outline tab moved.

Configuring Browser Development Tools

After you have Aptana Studio set up, you are ready to configure your browsers to debug JavaScript. In this section, you follow the steps needed to enable JavaScript debugging on each of the three main browsers. It doesn't really matter which browser you choose; however, Firefox is used in the next section and throughout the book.

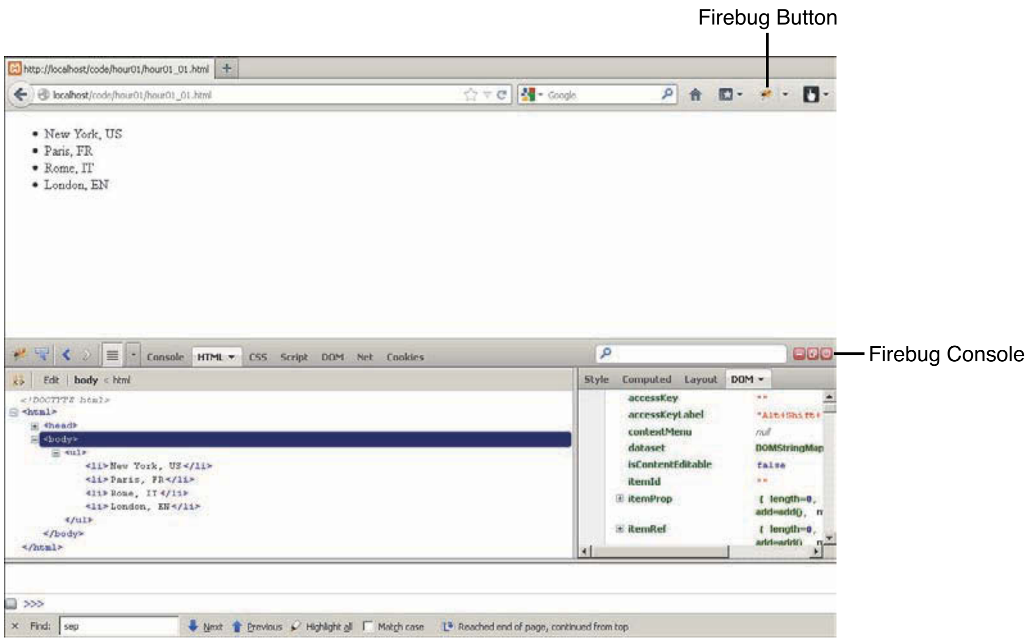
Firefox seems to have the most consistent experience and has been the most reliable. Firefox also seems to have the most consistent cross-platform support.

Installing Firebug on Firefox

Use the following steps to enable JavaScript debugging on Firefox:

1. Open Firefox.
2. Select Tools, Add-Ons from the main menu.
3. Type **Firebug** into the search box in the top right to search for Firebug, and then click the Install button to install it.
4. Type **FireQuery** into the search box in the top right to search for FireQuery, and then click the Install button to install it. FireQuery extends Firebug to also support jQuery.

5. When you reload Firefox, click the Firebug button to display the Firebug Console, as shown in Figure 1.10.



- New York, US
- Paris, FR
- Rome, IT
- London, EN

```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <ul>
        <li>New York, US</li>
        <li>Paris, FR</li>
        <li>Rome, IT</li>
        <li>London, EN</li>
    </ul>
  </body>
</html>
```

```
accessKey          ""
accessKeyLabel     "Alt+Shift+
contextMenu        null
dataset            DOMStringMap
isContentEditable  false
itemId             ""
itemProp           { length=0,
                   add=add(), r
itemRef            { length=0,
                   add=add(), r
```

```
>>>
```

FIGURE 1.10
Firefox with Firebug enabled.

Enabling Developer Tools in Internet Explorer

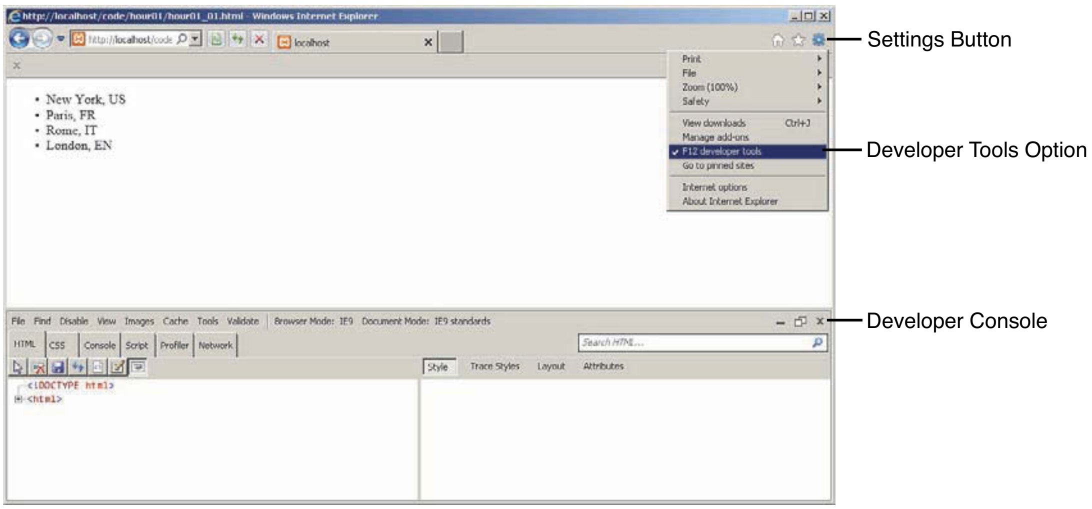
Use the following steps to enable JavaScript debugging on Internet Explorer:

1. Open Internet Explorer.
2. Click the Settings button and select Developer Tools from the drop-down menu, or press the F12 key.
3. The Developer Console is displayed as shown in Figure 1.11.

Enabling the JavaScript Console in Chrome

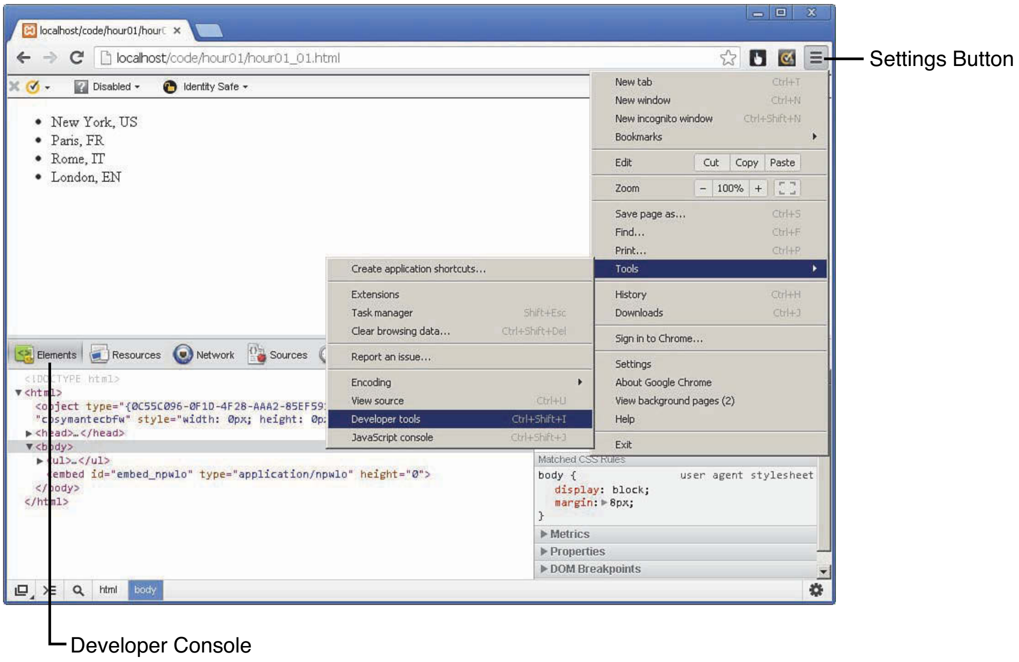
Use the following steps to enable JavaScript debugging in Chrome:

1. Open Chrome.
2. Click the Settings button and select Tools, Developer Tools from the drop-down menu. You can also press Ctrl+Shift+j on PCs or CMD-Shift-j on Macs.
3. The JavaScript Console is displayed as shown in Figure 1.12.



```
<!DOCTYPE html>
<html>
```

FIGURE 1.11
Internet Explorer with the Developer Console loaded.



```
<!DOCTYPE html>
▼<html>
   <object type="{0C55C096-0F1D-4F28-AAA2-85EF59
   "cpsymantecbfw" style="width: 0px; height: 0p
 ▶<head>…</head>
 ▼<body>
   ▶<ul>…</ul>
    <embed id="embed_npwlo" type="application/npwlo" height="0">
   </body>
</html>
```

```
body {                    user agent stylesheet
  display: block;
  margin: ▶ 8px;
}
```

FIGURE 1.12
Chrome with the Developer Console loaded.

Installing a Simple Development Web Server

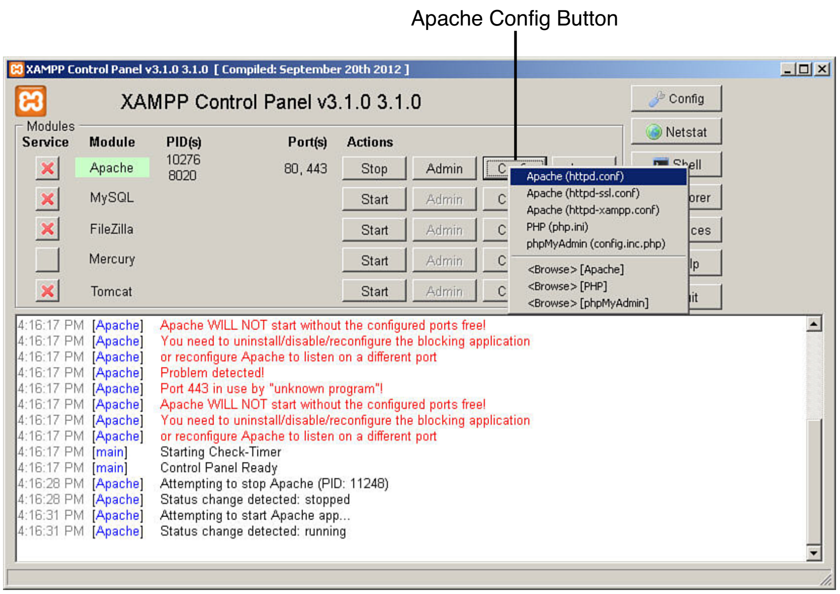
After you have your browser ready, the final step is to install and configure a simple development server. If possible, it is usually best to have a basic web server installed on your development machine.

You can choose from several options, but the two most common are Apache or IIS. The best option for a development server is using a prebuilt Apache stack that includes MySQL and PHP support. This book uses the XAMPP stack because it is available for Mac, Windows, and Linux.

The XAMPP stack can be downloaded from the following location:
www.apachefriends.org/en/xampp.html

Use the following steps to install and configure XAMPP as your development server:

1. Download the XAMPP installer and install XAMPP. The installation is straightforward. Remember the location where you choose to install it. You will be using that location later.
2. Load the XAMPP Control Panel shown in Figure 1.13.



```
4:16:17 PM  [Apache]   Apache WILL NOT start without the configured ports free!
4:16:17 PM  [Apache]   You need to uninstall/disable/reconfigure the blocking application
4:16:17 PM  [Apache]   or reconfigure Apache to listen on a different port
4:16:17 PM  [Apache]   Problem detected!
4:16:17 PM  [Apache]   Port 443 in use by "unknown program"!
4:16:17 PM  [Apache]   Apache WILL NOT start without the configured ports free!
4:16:17 PM  [Apache]   You need to uninstall/disable/reconfigure the blocking application
4:16:17 PM  [Apache]   or reconfigure Apache to listen on a different port
4:16:17 PM  [main]     Starting Check-Timer
4:16:17 PM  [main]     Control Panel Ready
4:16:28 PM  [Apache]   Attempting to stop Apache (PID: 11248)
4:16:28 PM  [Apache]   Status change detected: stopped
4:16:31 PM  [Apache]   Attempting to start Apache app...
4:16:31 PM  [Apache]   Status change detected: running
```

FIGURE 1.13
Selecting the Apache config file http.conf from the XAMPP Control Panel.

3. From the XAMPP Control Panel, click the Apache Config button and select the httpd.conf file to load the Apache configuration file in the editor.

4. Add the following directive to the httpd.conf file to enable a code directory for you to access directly when the server is running:

   ```
   <Directory "C:/xampp/htdocs/code">
       # Allows Browser Access to Your jQuery and JavaScript code Directory
       Options Indexes FollowSymLinks Includes ExecCGI
       AllowOverride All
       Allow from All
   </Directory>
   ```

CAUTION

The Allow from All option will allow anyone access to the files in that folder while the web server is running. It is perfect for debugging. If you would like stricter security, check out Apache's security guide at http://httpd.apache.org/docs/2.2/configuring.html.

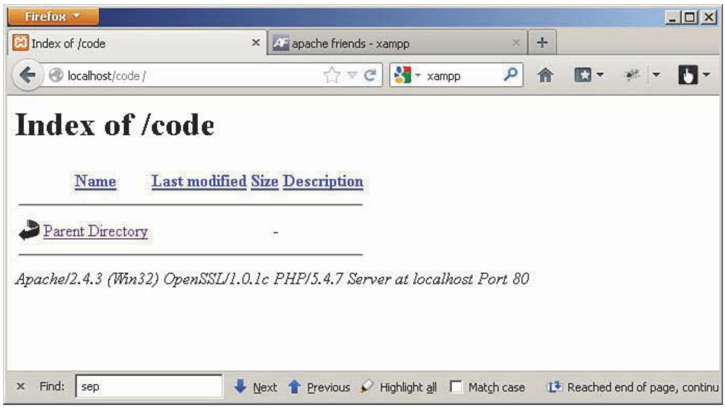
NOTE

In step 4, the directory is a Windows path for the default settings of XAMPP. If you installed XAMPP to a different location or on Linux and Mac systems, you will need to make adjustments to the path specified in the Directory directive.

5. Save the file.
6. Create a directory named code in the /xampp/htdocs directory or wherever your Apache root directory is set to. This is the directory that you will be adding your code to.
7. Stop and then Start the Apache service using the XAMPP Control Panel.
8. Go to the following location in your web browser; an empty directory link similar to the one in Figure 1.14 should be displayed in the browser:

   ```
   http://localhost/code
   ```

Your web server is now ready to be used for web development.



Index of /code

Name Last modified Size Description

Parent Directory -

Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7 Server at localhost Port 80

FIGURE 1.14
Verifying that the web browser has access to the newly created code directory.

Creating a Web Development Project

After you have installed your IDE and web server, you are ready to begin creating projects. In this section, you learn some concepts and go through the process of creating a project in Aptana Studio.

Directory Structure

When you first begin a web project, you typically start off small with a few images and only a couple of files. However, as time goes by and more files are added to the project, poorly organized projects can quickly become a mess.

To avoid that problem, plan your directory structure ahead of time. The best directory structure will depend on what your needs are, how many images you will be incorporating, what file types, and so on.

To give you a quick example, consider a basic directory structure similar to the following:

- **root**—Contains index.html, sitemaps, webcrawler items, and the like
- **root/html**—Contains only the HTML files
- **root/js**—Contains JavaScript files
- **root/php**—Contains any server-side PHP scripts
- **root/images**—Contains all graphics

- **root/images/visual**—Contains graphic elements, such as buttons, to build web pages
- **root/images/photos**—Contains any photos displayed on the website

The purpose of the preceding list is to give you an idea of one way to structure your files so that they remain organized. The best way is totally up to you. You may want more subdirectories; just don't add so many that the URL to reach files in them becomes a mess.

File Naming

Another area you need to pay attention to when creating a web project are filenames. Here are a few things to consider:

- **Not too long**—Filenames often become part of URLs and are parsed by JavaScript. Making filenames too long becomes cumbersome in code and in the browser.
- **Make them mean something**—When you create a script or HTML page, you will have to use that name when building web pages, and you will also need to find it in the editor. If the name doesn't reflect the purpose of the file, it can make development difficult.

TRY IT YOURSELF ▼

Creating a Project

In this section, you learn the process of creating a project in Aptana Studio. A project in Aptana Studio—and most IDEs, for that matter—is a way to organize, control, build, and often deploy websites and applications that require several files.

This section does not spend a lot of time discussing projects, but as you go along, you will get the idea of how a basic web project works.

Use the following steps to create a project in Aptana Studio:

1. Select File, New, Web Project to launch the Project Template dialog.
2. Select Default Project and click Next.
3. Type in a project name. Keep it short, but make it mean something. For example, TYjQueryCode is the project name for this book.
4. Unselect the Use Default Location option.
5. Add the location of the directory that you added to the httpd.conf file previously. In this case, it was c:\xampp\htdocs\code.
6. Click the Finish button and the project should show up in the workspace tab, as shown in Figure 1.15.

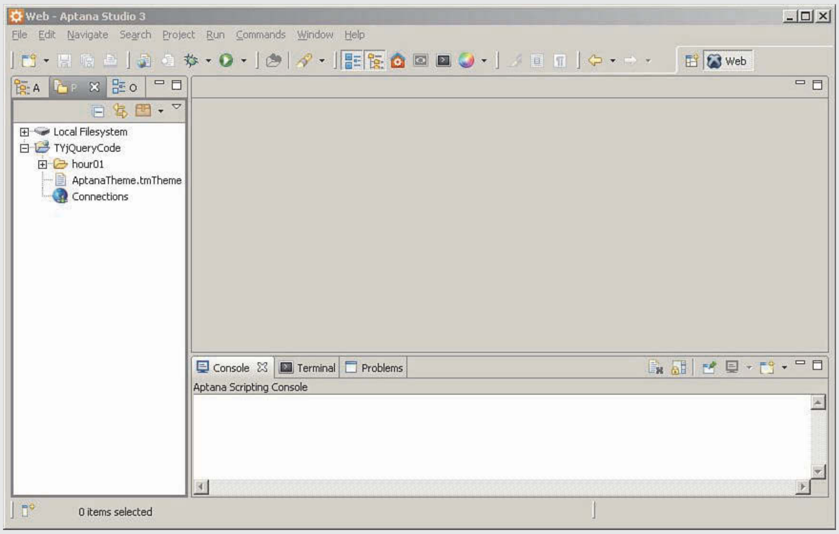


FIGURE 1.15
Creating a new project in Aptana Studio.

7. Right-click the project name and select New, Folder; then name the folder hour01 to store code for this hour.
8. Your first project has now been created.

TRY IT YOURSELF

Creating a Dynamic Web Page with jQuery and JavaScript

Now that you have a project created, you are ready to create your dynamic web pages. In this section, you follow the steps to create a fairly basic dynamic web page. When you are finished, you will have a dynamic web page based on HTML, stylized with CSS with interaction through jQuery and JavaScript.

Adding HTML

The first step is to create a simple web page that has an HTML element that you can stylize and manipulate. Use the following steps in Aptana to create the HTML document that you will use as your base:

1. Right-click the hour01 folder that you created earlier.
2. Select New, File from the pop-up menu.

3. Name the file first.html and click OK. A blank document should be opened up for you.
4. Type in the following HTML code. Don't worry if you are not too familiar with HTML; you'll learn enough to use it a bit later in the book.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <span>Click Me</span>
  </body>
</html>
```

5. Save the file.
6. Open the following URL in your web browser and you should see the text "Click Me" appear:

```
http://localhost/code/hour01/first.html
```

That's it. All the basic HTML elements are now in place. In the next section, you stylize the `<span>` element so that Click Me looks more like a button.

Adding CSS

The simple text rendered by the browser is pretty plain, but that problem can quickly be solved by adding a CSS style. In this section, you use CSS to make the text appear more like a button.

Use the following steps to add the CSS style to the `<span>` element. For reference, the style changes you make in these steps are shown in the final script in Listing 1.7:

1. Add the following code inside the `<head>` tags of the web page to include a CSS `<style>` element for all `<span>` elements:

```
<style>
  span{
  }
</style>
```

2. Add the following property setting to the span style to change the background of the text to a dark blue color:

```
background-color: #0066AA;
```

3. Add the following property settings to the span style to change the font color to white and the font to bold:

   ```
   color: #FFFFFF;
   font-weight: bold;
   ```

4. Add the following property settings to the span style to add a border around the span text:

   ```
   border-color: #C0C0C0;
   border:2px solid;
   border-radius:5px;
   padding: 3px;
   ```

5. Add the following property settings to the span style to set an absolute position for the span element:

   ```
   position:absolute;
   top:150px;
   left:100px;
   ```

6. Save the file.
7. Open the following URL in your web browser, and you should see the stylized text Click Me appear as shown in Figure 1.16:

   ```
   http://localhost/code/hour01/first.html
   ```

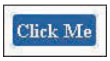


FIGURE 1.16
`<span>` element stylized to look like a button.

Writing a Dynamic Script

Now that the HTML is stylized the way you want it, you can begin adding dynamic interactions. In this section, you add a link to a hosted jQuery library so that you will be able to use jQuery, and then you link the browser mouse event `mouseover` to a JavaScript function that moves the text.

Follow these steps to add the jQuery and JavaScript interactions to your web page:

1. Change the `<span>` element to include an ID so that you can reference it, and also add a handler for the `mouseover` event, as shown in line 30 of Listing 1.7:

   ```
   <span id="elusiveText" onmouseover="moveIt()">Click Me</span>
   ```

2. Add the following line of code to the `<head>` tag, as shown in line 6 of Listing 1.7. This loads the jQuery library from a hosted source:

   ```
   <script src="http://code.jquery.com/jquery- latest.min.js"></script>
   ```

3. Add the following JavaScript function to the `<head>` as shown in lines 6–13 of Listing 1.7. This function creates an array of coordinate values from 10 to 350, then randomly sets the top and left CSS properties of the span element each time the mouse is moved over it:

   ```
   function moveIt(){
     var coords = new Array(10,50,100,130,175,225,260,300,320,350);
     var x = coords[Math.floor((Math.random()*10))];
     var y = coords[Math.floor((Math.random()*10))];
     $("#elusiveText").css({"top": y + "px", "left": x + "px"})
   }
   ```

4. Save the file.
5. Open the following URL in your web browser, and you should see the stylized text Click Me appear, as shown in Figure 1.16:

   ```
   http://localhost/code/hour01/first.html
   ```

6. Now try to click the Click Me button. The button should move each time the mouse is over it, making it impossible to click it.
7. Find someone who annoys you, and ask them to click the button.

LISTING 1.7 **A Simple Interactive jQuery and JavaScript Web Page**

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
05     <script src="http://code.jquery.com/jquery-latest.min.js"></script>
06     <script>
07       function moveIt(){
08         var coords = new Array(10,50,100,130,175,225,260,300,320,350);
09         var x = coords[Math.floor((Math.random()*10))];
10         var y = coords[Math.floor((Math.random()*10))];
11         $("#elusiveText").css({"top": y + "px", "left": x + "px"})
12       }
13     </script>
14     <style>
15       span{
16         background-color: #0066AA;
17         color: #FFFFFF;
18         font-weight: bold;
```

```
          border-color: #C0C0C0;
          border:2px solid;
          border-radius:5px;
          padding: 3px;
          position:absolute;
          top:150px;
          left:100px;
        }
      </style>
    </head>
    <body>
      <span id="elusiveText" onmouseover="moveIt()">Click Me</span>
    </body>
  </html>
```

Summary

In this hour, you learned the basics of web server and browser communications. You learned differences between GET and POST requests, as well as the purposes of server-side and client-side scripts. You also learned about the DOM and how the browser uses it to render the web page that is displayed to the user.

You have set up a good web development environment and created your first project. As part of creating your first project, you created a dynamic web page that incorporates HTML, CSS, jQuery, and JavaScript.

Q&A

Q. Which is better—a client-side or a server-side script?

A. It really depends on what you are trying to accomplish. Some people say that one way or the other is the only way to go. In reality, it is often a combination of the two that provides the best option. A good rule to follow is that if the interaction with the data is heavier based on user interaction such as mouse clicks, use a client-side script. If validation or error handling of the data requires interaction with the server, use a server-side script.

Q. Why don't all browsers handle JavaScript the same way?

A. To render HTML and interact with JavaScript, the browsers use an engine that parses the data from the server, builds objects, and then feeds them into a graphical rendering engine that writes them on the screen. Because each browser uses a different engine, each interprets the scripts slightly differently, especially with fringe elements that have not yet become standardized. If you want to support all browsers, you need to test your web pages in each of them to verify that they work correctly.

Workshop

The workshop consists of a set of questions and answers designed to solidify your understanding of the material covered in this hour. Try answering the questions before looking at the answers.

Quiz

1. Would you send a GET or a POST request to a web server to open a web page?
2. What type of script has access to browser mouse events: server-side, client-side, or both?
3. True or False: JavaScript consoles are enabled by default on all browsers.
4. What type of script is the best to use when defining the appearance of DOM elements?

Quiz Answers

1. GET
2. Client-side
3. False. You must manually enable JavaScript debugging on all browsers.
4. CSS scripts are the simplest to use when defining the appearance of DOM elements.

Exercises

1. Modify your first.html file to change the background color of your button randomly each time it is moved. Add the following two lines to randomly select a color:

   ```
   var colors = new Array("#0066AA", "#0000FF", "#FF0000", "#00FF00");
   var color = colors[Math.floor((Math.random()*4))];
   ```

   Then modify the CSS change in your JavaScript to include background-color, as shown next:

   ```
   $("#elusiveText").css({"top": y + "px", "left": x + "px", "background-color":
   ➥color})
   ```

2. Add an additional `<span>` element to your first.html file with the same behavior as the first. To do this, add the following two lines in the appropriate locations. You should be able to figure out where they go:

   ```
   $("#elusiveText2").css({"top": x + "px", "left": y + "px"})
   <span id="elusiveText2" onmouseover="moveIt()">Click Me</span>
   ```

This page intentionally left blank

HOUR 2

Debugging jQuery and JavaScript Web Pages

What You'll Learn in This Hour:

- ▶ Where to find information that is outputted from jQuery and JavaScript scripts
- ▶ How to debug problems with HTML elements
- ▶ Ways to more easily find and fix problems with CSS layout
- ▶ Methods to view and edit the DOM live in the web browser
- ▶ How to quickly find and fix problems in your JavaScript
- ▶ What information is available to analyze network traffic between the browser and the web server

A major challenge when writing JavaScript and jQuery applications is finding and fixing problems in your scripts. Simple syntax problems or invalid values can cause a lot of frustration and wasted time. For that reason, some excellent tools have been created to help you quickly and easily find problems in your scripts. In this hour, you learn some of the basics of debugging JavaScript via Firebug in Mozilla. Although the developer consoles in other browsers are a bit different, most of the principles are the same. Also, don't be alarmed if you don't recognize the code element in the examples. They'll be covered in upcoming hours, but you should be able to debug before you jump into coding heavily.

Viewing the JavaScript Console

One of the first debugging tools that you will want to become familiar with is the JavaScript console. The console is your interface to output from JavaScript scripts. Errors and log messages will be displayed as they occur in the JavaScript console.

For example, when an error in the script results in the browser not being able to parse it, the error will be displayed in the console. In addition to errors, by using the `console.log` statement, you can also add your own debug statements to be displayed in the JavaScript console.

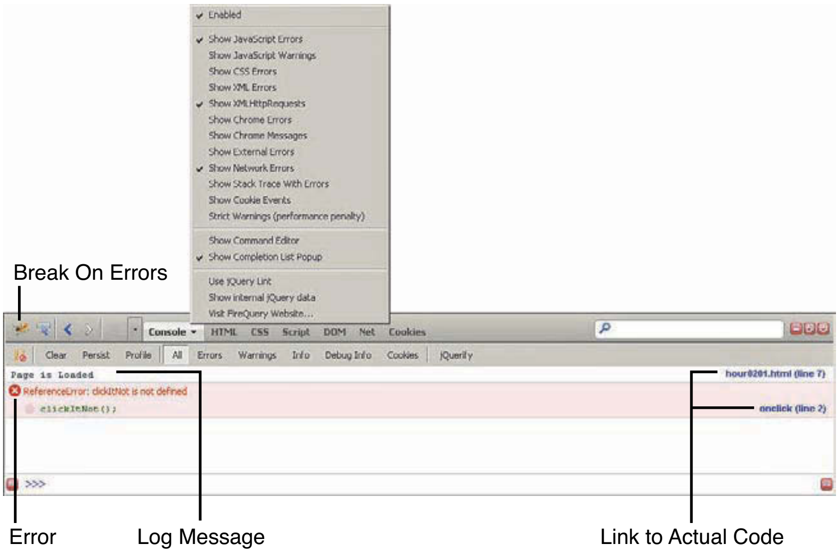
NOTE

In addition to `console.log`, you can use `console.error()`, `console.assert()`, and a variety of other statements to log information to the JavaScript console. For more information about how to use the Firebug console log, see

https://getfirebug.com/wiki/index.php/Console_API

Understanding the JavaScript Console

The JavaScript console is a fairly basic and yet powerful tool. The console has two parts: the controls and the list of log entries. Figure 2.1 shows the Firebug JavaScript console.



```
Page is Loaded                                            hour0201.html (line 7)
ReferenceError: clickItNot is not defined
    clickItNot();                                         onclick (line 2)
>>>
```

FIGURE 2.1
The JavaScript console in Firebug displays log messages and errors.

Notice the menu displayed when you click the down arrow in the Console tab. From that menu you can enable the console, as well as select which types of errors and log messages to include in the message list.

The console also provides a toolbar with several options. The options in the console toolbar are toggled by clicking them. The following list describes each of the options in the control bar:

- **Break On Errors**—When this is enabled, JavaScript will stop executing if an error is encountered in the script. This is very useful if you want to catch errors and see what the values of things are when they occur.

- **Clear**—Clears the messages in the message list.
- **Persist**—Retains the messages even if the page is reloaded. If this option is not set, the message list is emptied when the page is reloaded.
- **Profile**—Starts and stops the profiler to track time inside code.
- **All**—Displays all messages. For the most part, you should leave all messages on unless there are too many and you want to focus on a specific message.
- **Error**—Display only error messages.
- **Warnings**—Display only warning messages.
- **Debug Info**—Display only debug messages.
- **Cookies**—Display only cookie-related messages.
- **jQuerify**—Modifies the script that loads the jQuery library to include the latest jQuery code. This is part of the FireQuery plug-in.

Notice that in the messages portion in Figure 2.1, there are two types of messages. One is a log statement, and the second is an error. Both show the line number to the right. If you click the line number, you go directly to the code.

Notice in the error message, the top portion of text refers to the error that occurred and the bottom shows the actual JavaScript line. This is useful when debugging because you can often see the problem by looking at the error and the single line of code.

TRY IT YOURSELF ▼

Using the JavaScript Console to Find Errors

The simplest way to understand using the console is to debug an actual script. Consider the HTML code in Listing 2.1, which contains several errors. Use the following steps to add the listing to your project in Aptana:

1. Right-click the project and select New, Folder from the menu.
2. Name the folder hour02 and click Finish.
3. Right-click the new folder and select New, File from the menu.
4. Name the file hour0201.html.
5. Type in the contents of Listing 2.1, or if you have the file from the website, cut and paste the contents into the new file.
6. Save the file.

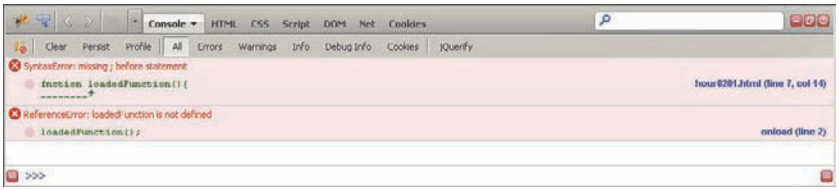
LISTING 2.1 A Very Simple HTML Document with JavaScript Errors

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <script>
      fnction loadedFunction(){
        console.log("Page is Loaded");
      }
      function clickIt(){
        console.log("User Clicked");
      }
    </script>
  </head>
  <body onload="loadedFunction()">
    <span onclick="clickItNot()">Click Me</span>
  </body>
</html>
```

The code in Listing 2.1 is supposed to display the message Page Is Loaded in the console after the page has been loaded in the browser. Another message, User Clicked, is displayed each time the user clicks the Click Me text in the browser. The problem is that the script has several bugs.

With the file now in place, use the following steps to debug the errors using the JavaScript console:

1. Open Firefox and click the Firebug icon.
2. Click the Console tab in Firebug to bring up the JavaScript console shown in Figure 2.2.



```
SyntaxError: missing ; before statement
  fnction loadedFunction(){                    hour0201.html (line 7, col 14)
  --------^
ReferenceError: loadedFunction is not defined
  loadedFunction();                            onload (line 2)
>>>
```

FIGURE 2.2
The JavaScript console showing two errors that occurred during the page load.

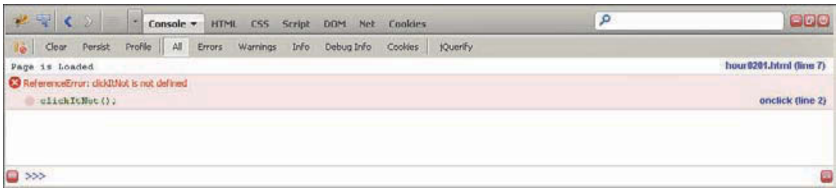
3. Open the following URL in Firefox to load the newly created web page:

   `http://localhost/code/hour02/hour0201.html`

4. Notice the errors displayed in the console, as shown in Figure 2.2. The first error shows that missing ";" in the definition for `loadedFunction()`. The second error shows that

loadedFunction is not defined. Taking these two errors together indicates that a problem exists with the definition for loadedFunction(). Looking at the failed definition statement, you can see that function is misspelled as fnction.

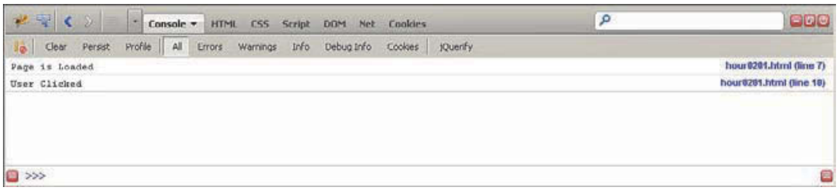
5. In Aptana, change the word fnction in line 6 to function.
6. Go back to Firefox and refresh the web page. Now in the console you should see Page Is Loaded, the text that is logged in the loadedFunction() function, but no errors.
7. Click the Click Me text. An error is added to the console, as shown in Figure 2.3. The error states that clickItNot is not defined. When you look at the HTML file and search for clickItNot, you can see on line 16 that an onclick event is linked to clickItNot(), but that the JavaScript function is named clickIt().



```
Page is Loaded                                   hour0201.html (line 7)
ReferenceError: clickItNot is not defined
    clickItNot();                                onclick (line 2)
```

FIGURE 2.3
The JavaScript console showing one successful log message and one error.

8. In Aptana, change clickItNot in line 15 to clickIt and save the file.
9. Reload the web pages.
10. Click the Click Me Text again. Figure 2.4 shows that both log statements are now displayed correctly and there are no errors. The page has been successfully debugged.



```
Page is Loaded                                   hour0201.html (line 7)
User Clicked                                     hour0201.html (line 10)
```

FIGURE 2.4
The JavaScript console showing two successful log messages and no errors.

Debugging HTML Elements

Debugging HTML elements can be a big challenge at times. Simple syntax errors can lead to major problems for the browser when it's trying to render an HTML document. In addition, HTML elements have property values that are not rendered to the screen but that will affect the behavior of the web page.

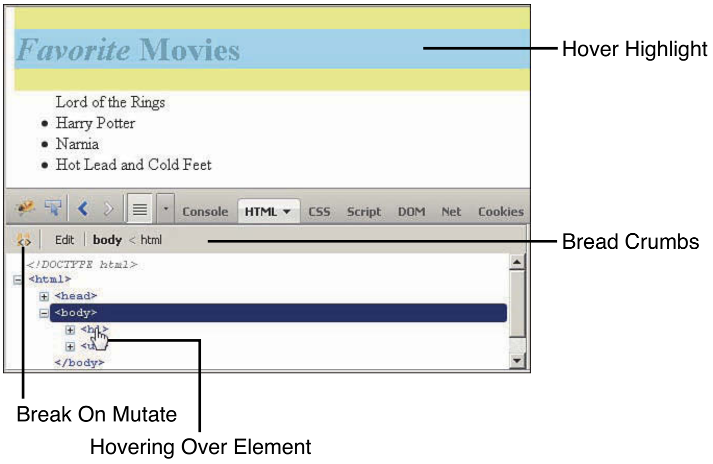
The HTML Inspector and the DOM editor help you find and fix problems in your HTML code. The following sections take you through some simple examples of using those tools.

Inspecting HTML Elements

The HTML Inspector enables you to view each of the HTML elements that have been parsed by the browser. This gives you a view of the HTML from the browser's perspective, which in the case of syntax errors is usually different from the one that was intended, making it more obvious where syntax errors are.

Figure 2.5 shows an example of the Firebug HTML Inspector. With the HTML Inspector, some very useful features are available to you as described next:

- **DOM Tree**—This is a simple view into the DOM tree. You can click the + icons to expand parts of the tree and click – icons to collapse parts of the tree.



```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <h1>
    <ul>
  </body>
```

FIGURE 2.5
The HTML Inspector page in Firebug.

- **Break on Mutate**—When this option is enabled, the browser will break into the JavaScript debugger whenever the DOM element is changed dynamically. This helps you catch problems as they are occurring.
- **Edit**—When this option is enabled, the tree view changes to a text editor view that allows you to directly edit the HTML code in the browser. The browser changes what is rendered based on the changes you make here. Although this won't change the code in your project, it is much easier to use this feature to try things out until problems are fixed. Then you can copy the code from the editor and paste it into the actual file in your project.
- **Hover**—When you hover over the HTML code in the DOM tree, the element is highlighted in the browser. The hover feature of the HTML Inspector is one of my favorites because it gives a very visual way to see the relationship between the node in the DOM tree and the rendered web page. Notice in Figure 2.5 that as the `<h1>` element hovered, the heading is highlighted in the web page.

NOTE

When an element is hovered over in the DOM tree, the element is highlighted on the web page. The hover highlight is color coded, with light blue being the contents, purple being the padding, and yellow being the margin for the HTML element.

- **Bread Crumbs**—The bread crumbs show the hierarchy of nodes from the root `<html>` node down to the one that is currently selected in the tree or edit view. This makes it easy to navigate around, especially in the edit view.

TRY IT YOURSELF ▼

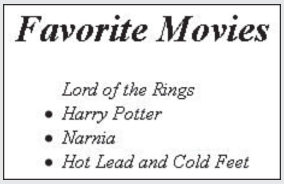
Debugging HTML Using the HTML Inspector

To illustrate how to use the HTML Inspector, consider the code in Listing 2.2. A basic HTML document with a list of movies and the word "Favorite" in the heading is supposed to be in italic. However, look at the rendered version in Figure 2.6. There are obviously some problems: Everything is in italic and there is no bullet point on the first list item. These problems are caused by just two characters in all the text.

LISTING 2.2 A Very Simple HTML Document with Some HTML Syntax Errors Illustrated in Figure 2.6

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

```
05   </head>
06   <body>
07     <h1><i>Favorite<i> Movies</h1>
08     <ul>
09       <ll>Lord of the Rings</li>
10       <li>Harry Potter</li>
11       <li>Narnia</li>
12       <li>Hot Lead and Cold Feet</li>
13     </ul>
14   </body>
15 </html>
```



Favorite Movies

Lord of the Rings
- *Harry Potter*
- *Narnia*
- *Hot Lead and Cold Feet*

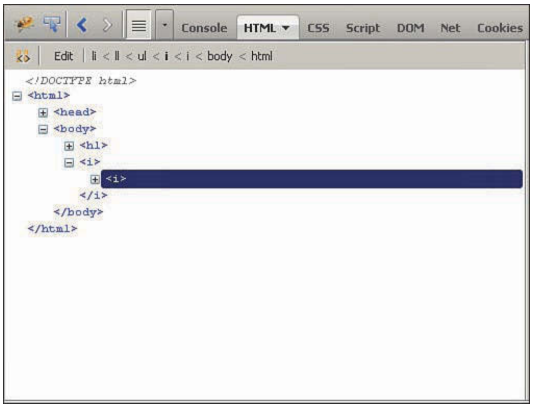
FIGURE 2.6
This web page has two problems: Only the word "Favorite" should be in italic, and there is no bullet point on the first list item.

Follow along with these steps to find and fix the HTML syntax problems using the HTML Inspector:

1. Add the code in Listing 2.2 to a new file hour0202.html in the hour02 folder of your project and save the document. You should be familiar with this process by now.
2. Open Firefox and click the Firebug icon to enable Firebug.
3. Open the following URL in Firefox; the web page should look like Figure 2.6.

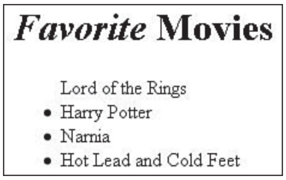
   `http://localhost/code/hour02/hour0202.html`

4. Click the HTML tab in Firebug and expand the `<html>`, then `<body>`, and then `<i>` tags, as shown in Figure 2.7. Notice that the only element under the `<i>` tag is a second `<i>` tag. That isn't right, so go back to Aptana and look at the `<i>` tags on line 7 in the HTML. Notice that the / is missing from the closing `<i>` tag.
5. Change the second `<i>` tag to a closing tag `</i>` and save the document.
6. Refresh the document in the browser. Notice that the word "Favorite" is now in italic, as it should be, but the bullet point is still missing, as shown in Figure 2.8.



```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <h1>
    <i>
      <i>
    </i>
  </body>
</html>
```

FIGURE 2.7
This HTML Inspector shows a second `<i>` in the DOM.

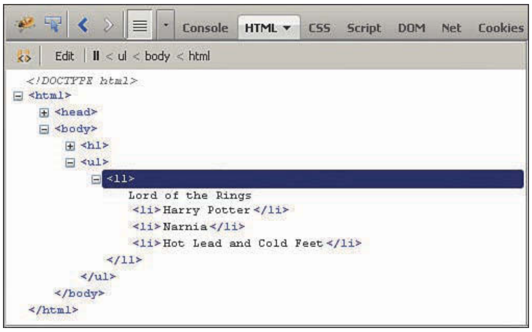


Favorite Movies

Lord of the Rings
- Harry Potter
- Narnia
- Hot Lead and Cold Feet

FIGURE 2.8
This web page now has only one problem—no bullet point on the first list item.

7. Go back to the HTML Inspector and expand the `<html>`, then `<body>`, then `<ul>`, then `<ll>`, as shown in Figure 2.9. Instead of a set of four `<li>` elements under the `<ul>` element, there is an `<ll>` element with the `<li>` elements underneath. We haven't covered the HTML tags yet, but if you are familiar with HTML lists, you will recognize that `ll` is not a valid HTML tag. It should be `<li>`.

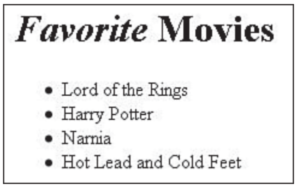


```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <h1>
    <ul>
      <ll>
        Lord of the Rings
        <li>Harry Potter</li>
        <li>Narnia</li>
        <li>Hot Lead and Cold Feet</li>
      </ll>
    </ul>
  </body>
</html>
```

FIGURE 2.9
Viewing the DOM reveals that the browser sees an `<ll>` tag under the `<ul>` tag, not a set of `<li>` tags.

8. Go back to Aptana and change the `<ll>` tag in line 9 to `<li>` and save the page.

9. Reload the web page in the browser. It is now displayed properly, as shown in Figure 2.10.



Favorite **Movies**

- Lord of the Rings
- Harry Potter
- Narnia
- Hot Lead and Cold Feet

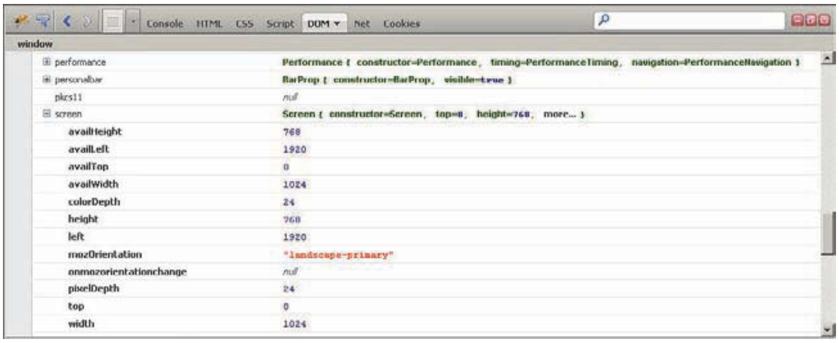
FIGURE 2.10
The properly formatted web page.

Viewing and Editing the DOM

Another important tool when debugging HTML is the DOM inspector. The DOM inspector is extremely powerful. It allows you to view the attributes, properties, functions, children, parents, and everything else about each HTML element in the DOM. The information is displayed in tree form so that you can expand and collapse groups.

The DOM inspector can be found in two places: either by clicking the DOM tab in Firebug or, when you are inspecting HTML, you can click the DOM tab in the HTML Inspector.

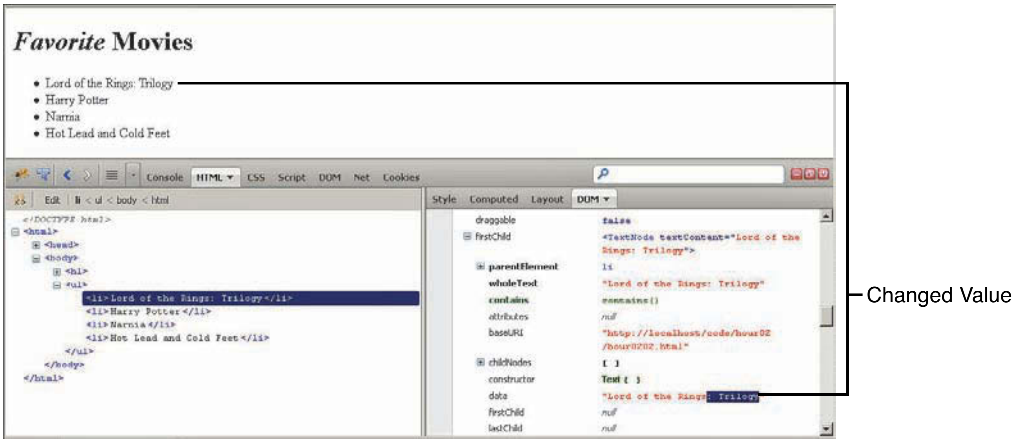
Figure 2.11 shows the main DOM inspector. From the main DOM inspector, you have access to a variety of information about the browser environment. For example, in Figure 2.11, the screen attribute of the window object is expanded, revealing the values of the available and actual dimensions of the browser window.



```
⊞ performance            Performance { constructor=Performance, timing=PerformanceTiming, navigation=PerformanceNavigation }
⊞ personalbar            BarProp { constructor=BarProp, visible=true }
  pkcs11                 null
⊟ screen                 Screen { constructor=Screen, top=0, height=768, more... }
    availHeight          768
    availLeft            1920
    availTop             0
    availWidth           1024
    colorDepth           24
    height               768
    left                 1920
    mozOrientation       "landscape-primary"
    onmozorientationchange null
    pixelDepth           24
    top                  0
    width                1024
```

FIGURE 2.11
The main DOM inspector tab in Firebug.

Typically, it's preferable to use the DOM inspector from the HTML Inspector, as shown in Figure 2.12. When you use the DOM tab in the HTML Inspector, you see only the DOM for that HTML element, which reduces the amount of information that is displayed. It also makes it easy to quickly change attribute values of the HTML element directly in the browser, which makes debugging and developing much easier.



- Lord of the Rings: Trilogy
- Harry Potter
- Narnia
- Hot Lead and Cold Feet

```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <h1>
    <ul>
      <li>Lord of the Rings: Trilogy</li>
      <li>Harry Potter</li>
      <li>Narnia</li>
      <li>Hot Lead and Cold Feet</li>
    </ul>
  </body>
</html>
```

```
draggable        false
firstChild       <TextNode textContent="Lord of the Rings: Trilogy">
  parentElement  li
  wholeText      "Lord of the Rings: Trilogy"
  contains       contains()
  attributes     null
  baseURI        "http://localhost/code/hour02/hour0202.html"
  childNodes     [ ]
  constructor    Text { }
  data           "Lord of the Rings: Trilogy"
  firstChild     null
  lastChild      null
```

FIGURE 2.12
Editing HTML elements inside the DOM inspector.

TRY IT YOURSELF ▼

Editing HTML Element Values in the DOM Inspector

As an example, you can play with the previous example of code using the following steps:

1. Open the fixed code in file hour0202.html in Firefox and open Firebug.
2. Click the HTML tab in Firebug.
3. Expand the `<html>`, `<body>`, and `<ul>` nodes.
4. Select the first `<li>` node.
5. Click the DOM tab to the right, as shown in Figure 2.12.
6. Scroll down and find the `firstChild` node in the DOM inspector and expand that node. It should be a `<TextNode>` element.
7. Double-click the value to the right of the data attribute and change the text as shown in Figure 2.12. Notice that the HTML element rendered in the web page also changes. It is as easy as that to manipulate any editable attribute of your HTML nodes.

Debugging CSS

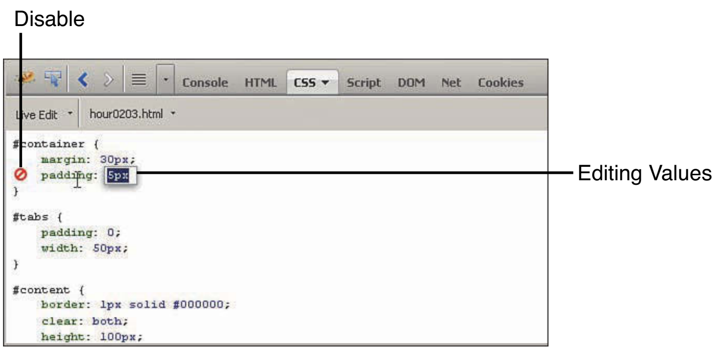
As part of debugging your dynamic web pages, you also need to be aware of how to debug CSS issues because a lot of the dynamics of web pages deal with modifying CSS layout in the JavaScript.

If your JavaScript or jQuery scripts modify the CSS layout of DOM elements, looking at the code in the web browser will not do you any good. You need to be able to see what CSS the browser has applied to the element. To do this, you need to use a combination of the CSS inspector as well as the layout inspector and style inspector inside the HTML Inspector.

Using the CSS Inspector

The CSS inspector, shown in Figure 2.13, provides access to all the CSS scripts loaded in the web page. There are two drop-down menus at the top of the CSS inspector. The menu on the left allows you to toggle between the following options:

- **Source Edit**—Displays the CSS that originally loaded with the web page.
- **Live Edit**—Displays the CSS that is currently applied to the HTML elements.



```
#container {
    margin: 30px;
    padding: 5px
}

#tabs {
    padding: 0;
    width: 50px;
}

#content {
    border: 1px solid #000000;
    clear: both;
    height: 100px;
```

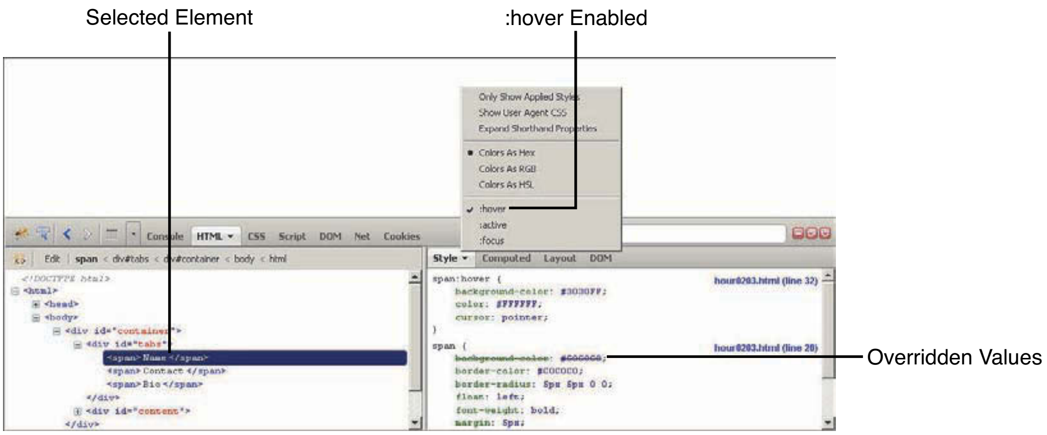
FIGURE 2.13
Editing CSS properties inside the CSS inspector.

The menu on the left provides a list of all the files containing CSS that have been loaded. This enables you to select which CSS document you would like to view and edit.

From the CSS inspector, you also have the capability to edit the CSS. Figure 2.13 shows the editing in process. Notice the disable icon. When you click this icon, that CSS property will be disabled, and the icon will go from red to gray. You can also directly edit the value of the CSS property, as shown in Figure 2.13.

Using the Style Inspector

In addition to editing the entire CSS file, you can view and edit the CSS properties for specific elements from the HTML Inspector. Figure 2.14 shows the Style tab in the HTML Inspector. From the Style inspector, you can view and modify the property values for a specific element.



```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <div id="container">
      <div id="tabs">
        <span>Name</span>
        <span>Contact</span>
        <span>Bio</span>
      </div>
    <div id="content">
  </div>
```

```
span:hover {                                  hour0203.html (line 32)
    background-color: #3030FF;
    color: #FFFFFF;
    cursor: pointer;
}
span {                                        hour0203.html (line 20)
    ~~background-color: #C0C0C0;~~
    border-color: #C0C0C0;
    border-radius: 5px 5px 0 0;
    float: left;
    font-weight: bold;
    margin: 5px;
```

FIGURE 2.14
Editing CSS properties inside the Style inspector inside the HTML Inspector.

Figure 2.14 also illustrates some important features of the Style inspector. Notice that `:hover` is selected in the menu. That shows the CSS style that is applied to that element when it is hovered over by the mouse. Also notice that the `span:hover` selector overrides the `background-color` setting in the `span` selector. The entire CSS hierarchy is displayed in the style window so you can see which property values are coming from what CSS selector and which values have been overridden.

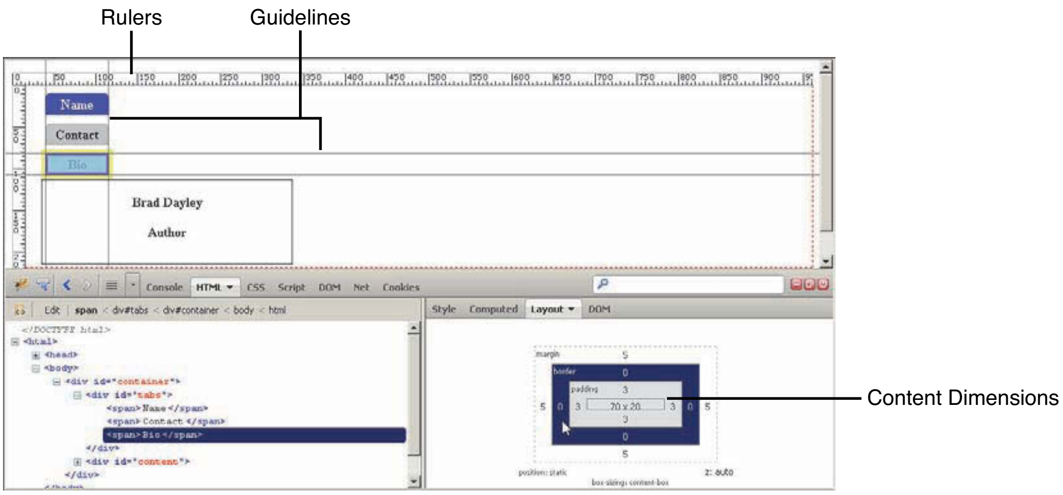
Using the Layout Inspector

Another extremely powerful tool when debugging CSS is the Layout inspector in the HTML Inspector. The Layout inspector, shown in Figure 2.15, provides an easy-to-use visual interface to the CSS layout of the selected HTML element.

From the Layout inspector you can use, view, and modify the following features:

- **Margin**—The margin is the outermost box shown in Layout inspector. There is a value on each of the four sides of the margin. You can double-click those values and change the CSS property directly in the Layout inspector.
- **Border**—The border is the next box. It also has four values you can change to adjust the CSS border properties of the HTML element.

- **Padding**—The padding is the next box. It also has four values you can change to adjust the CSS padding properties of the HTML element.
- **Content**—The content is the innermost box in the Layout inspector. It has two values, the length and width, that you change to set the CSS length and width properties of the HTML element.
- **Rulers**—The rulers are displayed in the web page to give you a specific size scale to work from.
- **Guidelines**—When you select the margin, border, padding, or content box in the Layout inspector, guidelines appear in the web page. The guidelines run horizontally and vertically to show the specific location of the edges of that CSS property. This can be extremely useful when trying to line up elements in your layouts.



```
</DOCTYPE html>
<html>
  <head>
  <body>
    <div id="container">
      <div id="tabs">
        <span>Name</span>
        <span>Contact</span>
        <span>Bio</span>
      </div>
      <div id="content">
    </div>
```

FIGURE 2.15
Viewing the CSS layout properties inside the Layout inspector inside the HTML Inspector.

▼ TRY IT YOURSELF

Editing the CSS Layout

To help you understand debugging and editing the CSS layout using Firebug, consider the code in Listing 2.3. The code is designed to display a simple tabbed box to display info. Some problems exist with the CSS properties that cause it to be displayed poorly, as shown in Figure 2.16. Notice that the tabs are stacked and there is space between them.

LISTING 2.3 A Very Simple HTML Document with Some HTML Syntax Errors Illustrated in Figure 2.16

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <style>
      #container{
        margin: 30px;
        padding:5px;
      }
      #tabs{
        padding: 0px;
        width:50px;
      }
      #content{
        border: 1px solid #000000;
        height: 100px;
        width: 300px;
        clear: both;
      }
      span{
        margin: 5px;
        width: 70px;
        background-color: #C0C0C0;
        font-weight: bold;
        border-color: #C0C0C0;
        border:1px solid, #000000;
        border-radius: 5px 5px 0px 0px;
        padding: 3px;
        float: left;
        text-align: center;
      }
      span:hover{
        background-color: #3030FF;
        color: #FFFFFF;
        cursor: pointer;
      }
      p{
        font-weight: bold;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <div id="container">
```