Brad Dayley

ESSENTIAL CODE AND COMMANDS

jQuery and JavaScript

PHRASEBOOK

jQuery and JavaScript

PHRASEBOOK

This page intentionally left blank

jQuery and JavaScript

PHRASEBOOK

Brad Dayley

Addison-Wesley

Upper Saddle River, NJ · Boston · Indianapolis · San Francisco
New York · Toronto · Montreal · London · Munich · Paris · Madrid
Cape Town · Sydney · Tokyo · Singapore · Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2013950281

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-91896-3
ISBN-10: 0-321-91896-7

First printing: December 2013

Acquisitions Editor
Mark Taber

Managing Editor
Kristy Hart

Project Editor
Katie Matejka

Copy Editor
Karen Gill

Indexer
Publishing Services, WordWise, Larry Sweazy

Proofreader
Kathy Ruiz

Technical Reviewer
Phil Ballard

Editorial Assistant
Vanessa Evans

Cover Designer
Chuti Prasertsith

Senior Compositor
Gloria Schurick

Dedication

For D!
A & F

Contents

1 Jumping into jQuery, JavaScript, and the World of Dynamic Web Development 1

Understanding JavaScript 2
Introducing jQuery 4
Introducing jQuery UI 7
Introducing jQuery Mobile 9
Configuring Browser Development Tools 12

2 Using the JavaScript Language 15

JavaScript Syntax 15
Defining and Accessing Data 16
Defining Functions 20
Manipulating Strings 21
Manipulating Arrays 25
Applying Logic 29
Math Operations 31
Working with Dates 36

3 Interacting with the Browser 43

Writing to the JavaScript Console 43
Reloading the Web Page 44
Redirecting the Web Page 44
Getting the Screen Size 45
Getting Current Location Details 45
Accessing the Browser 47
Using the Browser History to Go Forward and Backward Pages 49
Creating Popup Windows 50
Manipulating Cookies 52
Adding Timers 55

4 Accessing HTML Elements 59
Finding HTML Elements in JavaScript 59
Using the jQuery Selector to Find HTML Elements 61
Chaining jQuery Object Operations 75
Navigating jQuery Objects to Select Elements 76

5 Manipulating the jQuery Object Set 83
Getting DOM Objects from a jQuery Object Set 84
Converting DOM Objects into jQuery Objects 84
Iterating Through the jQuery Object Set Using `.each()` 85
Using `.map()` 87
Assigning Data Values to Objects 89
Adding DOM Elements to the jQuery Object Set 91
Removing Objects from the jQuery Object Set 91
Filtering the jQuery Object Results 92

6 Capturing and Using Browser and User Events 95
Understanding Events 96
Adding Event Handlers 99
Controlling Events 107
Using Event Objects 111
Handling Mouse Events 115
Handling Keyboard Events 118
Form Events 122

7 Manipulating Web Page Elements Dynamically 125
Getting and Setting DOM Element Attributes and Properties 126

Getting and Setting CSS Properties 130
Getting and Manipulating Element Content 139

8 Manipulating Web Page Layout Dynamically 143
Hiding and Showing Elements 143
Adjusting Opacity 146
Resizing Elements 149
Repositioning Elements 152
Stacking Elements 156

9 Dynamically Working with Form Elements 159
Getting and Setting Text Input Values 160
Checking and Changing Check Box State 161
Getting and Setting the Selected Option in a Radio Group 162
Getting and Setting Select Values 164
Getting and Setting Hidden Form Attributes 166
Disabling Form Elements 167
Showing/Hiding Form Elements 170
Forcing Focus to and Away from Form Elements 172
Controlling Form Submission 175

10 Building Web Page Content Dynamically 177
Creating HTML Elements Using jQuery 178
Adding Elements to the Other Elements 179
Removing Elements from the Page 184
Dynamically Creating a Select Form Element 186
Appending Rows to a Table 189
Inserting Items into a List 191
Creating a Dynamic Image Gallery 193
Adding HTML5 Canvas Graphics 196

11 Adding jQuery UI Elements 201
Adding the jQuery UI Library 201
Implementing an Autocomplete Input 203
Implementing Drag and Drop 205
Adding Datepicker Element 212
Using Sliders to Manipulate Elements 215
Creating a Menu 219
Adding Tooltips 223

12 Animation and Other Special Effects 227
Understanding jQuery Animation 228
Animating Visibility 234
Making an Element Slide Back to Disappear 238
Animating Show and Hide 242
Animating Resizing an Image 246
Animating Moving an Element 248

13 Using AJAX to Communicate with Web Servers and Web Services 251
Understanding AJAX 251
AJAX from JavaScript 261
AJAX from jQuery 267
Handling jQuery AJAX Responses 282
Using Advanced jQuery AJAX 285

14 Implementing Mobile Web Sites with jQuery 291
Getting Started with jQuery Mobile 291
Building Mobile Pages 302
Implementing Mobile Sites with Multiple Pages 306
Creating a Navbar 314
Applying a Grid Layout 316
Implementing Listviews 320

Using Collapsible Blocks and Sets 326
Adding Auxiliary Content to Panels 327
Working with Popups 329
Building Mobile-Friendly Tables 332
Creating Mobile Forms 334

Index 341

Acknowledgments

I'd like to take this page to thank all those who made this title possible. First, I thank 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; Karen Gill for turning the ramblings of my techie mind into coherent text; Phil Ballard for ensuring the accuracy in the book and keeping me honest; Kathy Ruiz and Gloria Schurick for making sure the book is the highest quality; Larry Sweazy for making sure that the readers can actually find what they look for in the book; Tammy Graham and Laura Robbins for their graphical genius; Chuti Prasertsith for the stylish and sleek cover; and Katherine Matejka for all her hard work in making sure this book is the best it can be. You guys are awesome!

About the Author

Brad Dayley is a senior software engineer with 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 Developer's Phrasebook* and *Sams Teach Yourself jQuery and JavaScript in 24 Hours*.

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.

You can email or write directly 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 stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and contact information.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our Web site 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

1

Jumping into jQuery, JavaScript, and the World of Dynamic Web Development

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, dynamic web development has changed forever.

This chapter focuses on providing you with some concepts of dynamic web programming and getting set up to use JavaScript and jQuery in your web pages.

Understanding JavaScript

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

This allows JavaScript code to dynamically 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 enables JavaScript to provide functionality such as dynamic lists as well as drag and drop. Figure 1.1 shows an example of downloading a web page from a server and then using JavaScript code to dynamically populate a `<ul>` element with `<li>` children.

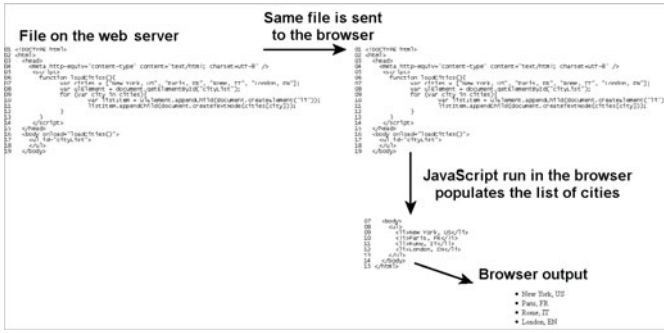


Figure 1.1 JavaScript runs in the browser and can change the HTML elements on the web page without needing to retrieve additional pages from the web server.

Adding JavaScript to an HTML Document

```
<head>
  <script type="text/javascript">
    alert("JavaScript is Enabled!");
  </script>
</head>
```

You can add JavaScript to an HTML document using HTML `<script>` tags. When the browser encounters a `<script>` tag, it parses the contents and then executes the JavaScript inside. Typically, the `<script>` tags are added to the `<head>` element, but you can also add them to the `<body>` element.

The web browsers currently default all scripts to `javascript`; however, it is a good idea to still set the type to `"text/javascript"` for future compatibility if that changes.

Watch out!

The order that `<script>` tags appear in the HTML document determines their load order. If you are loading multiple scripts, keep in mind that using the same variable and function names in subsequent scripts overwrites the ones already defined in previous ones.

Loading JavaScript from External Files

```
<head>
  <script type="text/javascript"
    src="scripts/scriptA.js"></script>
  <script type="text/javascript"
    src="scripts/scriptB.js"></script>
</head>
```

As you create more and more complex JavaScript web applications, you will find that adding the JavaScript to your HTML files doesn't make much sense. The files become too big, and you will often want to reuse the scripts in other web pages.

Instead of including the JavaScript inside the `<script>` tag, you can specify a `src` location for the script to be loaded from. When the browser encounters a `src` attribute in the `<script>` tag, it downloads the script from the server and loads it into memory.

Watch out!

The browser downloads the external scripts asynchronously. That means you need to be careful if you reference one script from another because the second script may not be downloaded yet.

Introducing jQuery

jQuery is a library that is built on JavaScript. The underlying code is actually 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 list and tables. Selectors 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 Document Object Model (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.

Loading jQuery in Your Web Pages

```
<head>
  <script src="local/jquery-2.0.3.min.js"></script>
. . . or . . .
  <script src=
    "http://code.jquery.com/jquery-2.0.3.min.js">
  </script>
</head>
```

The jQuery library is just a `.js` file. You can load it just like any other JavaScript file. There are two ways to add jQuery to your web page.

The easiest method of adding jQuery to web pages is to use one of the several Content Discovery Network locations, or CDNs. A CDN allows you to load the libraries from a network of jQuery hosting servers spread globally. The benefit of this method is that the servers are spread globally so the downloads are distributed to multiple servers. Also, if the user has loaded another web page with a link to the CDN file, it may already have the jQuery library cached. The following are some examples of hosting CDNs:

```
//jQuery
<script src=
  "http://code.jquery.com/jquery-2.0.3.min.js">
</script>
<script src=
  "http://code.jquery.com/jquery-migrate-
➥1.1.0.min.js">
</script>
//google
<script src=
  "https://ajax.googleapis.com/ajax/libs/jquery/
➥2.0.3/jquery.min.js">
</script>
//Microsoft
<script src=
  "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
➥2.0.3.min.js">
</script>
```

The other option is to download the `.js` file from http://jquery.com/download and load it with your other JavaScript libraries. This has the advantage of being more closely tied to your site's content so you don't have to worry about possible unavailability issues or site-blocking issues with the external locations.

Accessing jQuery in Your JavaScript

```
jQuery("#myElement")
. . . or . . .
$("#myElement")
```

jQuery is accessed using the jQuery object that is defined when the library is loaded. jQuery also provides a shortcut `$` character that you can use in the phrases throughout the book. For example, the following two jQuery statements are identical:

```
jQuery("#myElement")
$("#myElement")
```

Introducing jQuery UI

jQuery UI is an additional library built on top of jQuery. The purpose of jQuery UI is to provide a set of extensible interactions, effects, widgets, and themes that make it easier to incorporate professional UI elements in your web pages.

jQuery UI is made up of two parts: JavaScript and Cascading Style Sheets (CSS). The JavaScript portion of jQuery UI extends jQuery to add additional functionality specific to adding UI elements or applying effects to those elements. The CSS portion of jQuery UI styles the page elements so that developers don't need to style the elements every time.

jQuery UI saves developers time by providing prebuilt UI elements, such as calendars and menus, with interactions, such as dragging and resizing, right out of the box.

Getting the jQuery UI Library

To get started with jQuery UI, you need to download the library and add it to your web pages. You can download the jQuery library from http://jqueryui.com/download/ by selecting the options that you would like to include and clicking on the Download button at the bottom. This downloads the jQuery UI files.

When you download the jQuery UI library, you get a zip file. Inside that file are three main folders that you need to understand. They are

- **`js`**—This is the folder that contains the jQuery UI and jQuery libraries files. You need to deploy these files so you can load them in your web pages.
- **`css`**—This contains the theme-named folders that house the `.css` files and an `images` folder used by the jQuery UI library. The .css file and images/ folder need to be placed in the same location and accessible from your web pages.
- **`development-bundle`**—This folder contains the full source for jQuery UI. If you are not planning to modify the jQuery UI code, you can ignore this folder.

Loading jQuery UI

```
<head>
  <link rel="stylesheet" type="text/css"
    href="local/jquery-ui-1.10.3.css">
  <script src="local/jquery-2.0.3.min.js"></script>
  <script src="local/jquery-ui-
➥1.10.3.min.js"></script>
. . . or  for CDN. . .
  <link rel="stylesheet" type="text/css"
    href="http://code.jquery.com/ui/1.10.3/themes/
➥base/jquery-ui.css">
  <script src=
    "http://code.jquery.com/jquery-2.0.3.min.js">
  </script>
  <script src=
    "http://code.jquery.com/ui/1.10.3/jquery-ui.js">
  </script>

</head>
```

To load jQuery UI, you first need to load the jQuery library. The jQuery UI is also a `.js` file. You can load it just like any other JavaScript file. Also, just like jQuery, you can load the script from an external CDN source or download the library and load it locally.

You also need to load the jQuery UI `.css` file using a `<link>` tag. This can be a local file or an external CDN location. For example:

```
<link rel="stylesheet" type="text/css"
  href="local/jquery-ui-1.10.3.min.css">
. . . or  for CDN. . .
<link rel="stylesheet" type="text/css"
  href="http://code.jquery.com/ui/1.10.3/themes/
➥base/jquery-ui.css">
```

Introducing jQuery Mobile

Mobile devices are the fastest growing development platform. Much of that development is geared toward making web sites mobile friendly. It is much easier to implement and maintain a mobile web site than it is to maintain a mobile application.

jQuery mobile is an additional library built on top of jQuery. It is designed to streamline many of the necessary structural, UI, and event implementations to build mobile web sites. jQuery Mobile provides several advantages with developing mobile solutions, including

- Hiding some of the complexities of resizing page elements to a wide array of mobile devices.
- Providing simple UI components with mobile event interactions already built into them.

- Building on JavaScript and jQuery provides a common and well developed platform that is based on proven concepts.
- Developing a mobile web apps is simple to do and does not require any installation of the user's part. That is why they are becoming more and more popular.

Getting the jQuery Mobile Library

To get started with jQuery Mobile, you need to download the library and add it to your web pages. You can download the jQuery library from http://jquerymobile.com/download/. You can also download jQuery Mobile from http://jquerymobile.com/download-builder/ by selecting the version and options that you would like to include and clicking on the Download button at the bottom. This downloads a zip file containing the jQuery Mobile library.

Watch out!

The `.css` files and the `images` folder that are included with the jQuery Mobile library come as a set. You need to make sure they are installed in the same location and you don't mix and match them from different custom downloads.

When you download the jQuery Mobile library, you get a zip. Inside the zip file are three main components that you need to put where your mobile web pages can load them. They are

- **`js` files**—There will be a `jquery.moble.###.js` as well as a minified version. This is the main library file, and one of these needs to be placed where you can add it to your project files in a `<script>` tag.
- **`css` files**—There will be `jquery.mobile.###.css`, `jquery.mobile.###.structure.css`, and `jquery.mobile.###.theme.css` files as well as their minified forms. This is all the styling code and should be placed in the same location as the `jquery.moble.###.js` file.
- **`images` folder**—This folder contains the images and icons used by jQuery Mobile to style the elements. This should also be placed in the same location as the `jquery.moble.###.js` file.

Loading jQuery Mobile

```
<head>
  <link rel="stylesheet"
    href="local/jquery.mobile-custom.min.css" />
  <script src="local/jquery-2.0.3.min.js"></script>
  <script src="local/jquery.mobile-
➥custom.min.js"></script>
. . . or  for CDN . . .
  <link rel="stylesheet" href=
    "http://code.jquery.com/mobile/1.4.0/jquery.
➥mobile-1.4.0.min.css" />
  <script src=
   "http://code.jquery.com/jquery-
➥1.8.2.min.js"></script>
  <script src=
   "http://code.jquery.com/mobile/1.4.0/jquery.
➥mobile-1.4.0.min.js">
   </script>
</head>
```

Similar to what you did for jQuery UI, you need to load jQuery before loading jQuery Mobile. Once jQuery is loaded, you can load the jQuery Mobile `.js` file from an external CDN source or a locally downloaded version of the library.

You need to load the jQuery Mobile `.css` file as well using a `<link>` tag. For example, the following code loads the jQuery library first because it is required by jQuery Mobile and then loads the jQuery Mobile JS and CSS files:

```
<script type="text/javascript"
  src="local/jquery-2.0.3.min.js"></script>
<script type="text/javascript"
  src="local/jquery.mobile-custom.min.js"></script>
<link rel="stylesheet"
  href="local/jquery.mobile-custom.min.css" />
```

Configuring Browser Development Tools

An important aspect of developing JavaScript and jQuery is using the web development tools incorporated in web browsers. These tools allow you to see the script files loaded, set breakpoints, step through code, and much, much more. It is beyond the scope of this book to delve too much into the browser tools. However, I wanted to provide you with the steps to enable them and encourage you to learn about them if you have not already done so.

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` in the search box in the top right to search for Firebug. Then click the Install button to install it.
4. Type `FireQuery` in the search box in the top right to search for FireQuery. Then click the Install button to install it. FireQuery extends Firebug to also support jQuery.
5. When you reload Firefox, click on the Firebug button to display the Firebug console.

Enabling Developer Tools in Internet Explorer

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

1. Open IE.
2. Click on the Settings button and select Developer Tools from the drop-down menu. Or you can press the F12 key.
3. The Developer console is displayed.

Enabling the JavaScript Console in Chrome

Use the following steps to enable JavaScript debugging in Chrome:

1. Open Chrome.
2. Click on the Settings button and select Tools > Developer Tools from the drop-down menu. Or you can press Ctrl+Shift+J on PCs or Cmd+Shift+J on Macs.
3. The JavaScript console is displayed.

2

Using the JavaScript Language

The phrases in this chapter focus on various ins and outs of the JavaScript language. You need to know these to be able to fully utilize the full capabilities that jQuery and JavaScript provide in the HTML stack.

JavaScript Syntax

As a programming language, JavaScript, and consequently jQuery since it is based on JavaScript, requires a specific syntax. This section gives you a quick primer on the JavaScript syntax before going into the language details.

You should be familiar with the following rules regarding JavaScript syntax:

- All JavaScript statements end with a semicolon.
- Statements execute in the order they appear in the script.

- Blocks of code are defined using {CODE_BLOCK} brackets.
- Expressions bound in () brackets are evaluated before they're applied to the rest of the statement.
- JavaScript is case sensitive.
- To break a code line in the middle of a string, \ must be the last character on the line.
- Strings in JavaScript must be enclosed in either single ('string') or double ("string") quotes.
- When adding a string to a number, the resulting value is a string.
- Variables must begin with a letter, $, or _.

Defining and Accessing Data

One of the most important aspects of JavaScript is the ability to define variables that represent different forms of data, from individual numbers and strings to complex objects with properties and methods to arrays containing several items. Once data has been defined, you can use and reuse it for a variety of purposes. This section provides phrases that help you create the various data structures that you will need to use in JavaScript.

Defining and Accessing Variables

```
var x = y = 5;
var z = 10;
var results = "x + y + z = " + (x+y+z);
var s1 = "jQuery";
var s2 = "JavaScript"
var s3 = s1 + " & " + s2;
```

JavaScript makes it easy to define variables. Variables are assigned using `var name = value;` syntax. The `var` keyword tells JavaScript that this is a new variable name. Once the variable has been assigned a value, you can access it using the variable name. You can define multiple variables on a single line using `var name1 = name2 = value;` syntax.

When you assign a variable to an expression, such as `x + y + z`, the expression is evaluated before assigning the variable a value. The following code snippet shows you how to define and access different variables.

```
<script>
  var x = y = 5;
  var z = 10;
  var results = "x + y + z = " + (x+y+z);
  var s1 = "jQuery";
  var s2 = "JavaScript"
  var s3 = s1 + " & " + s2;
  document.write(results);
  document.write("<br>");
  document.write(s3);
</script>
```

ch0201.html

```
x + y + z = 20
jQuery & JavaScript
```

Output from ch0201.html

Creating Arrays

```
var x = y = 5;
var z = 10;
var results = "x + y + z = " + (x+y+z);
var s1 = "jQuery";
var s2 = "JavaScript"
var s3 = s1 + " & " + s2;
```

You can create arrays in a few different ways. You can create them in the definition using `[]` brackets, such as `var arr=[1,2,3]`. You can also create a new array object and add items to it using the `push()` method. You can access items in the array using the `arr[index]` method. The following code snippet shows an example of creating an array in each of these ways.

```
<script>
  var weekdays = ["Mon", "Tue", "Wed", "thur",
➥"Fri"];
  var weekend = new Array();
  weekend.push("Sat");
  weekend.push("Sun");
  document.write(weekdays.toString() + "<br>");
  document.write(weekend[0] + ", " + weekend[0]);
</script>
```

ch0202.html

```
Mon,Tue,Wed,thur,Fri
Sat,Sun
```

Output from ch0202.html

Creating Objects

```
  var obj1 = {name:"Brad", title:"Author", "last-
➥name":"Dayley" };
  var obj2 = new Object();
  obj2.name = "Frodo";
  obj2.title = "Hobbit";
  obj2["last-name"] = "Baggins";
```

You can create objects in the definition using `{property:value, ..}` syntax, such as `var obj={name:"Brad", title:"Author"};`. You can also create a new object and add properties to it using standard dot syntax.

The following code snippet shows an example of creating an array in each of these ways.

```
<script>
  var obj1 = {name:"Brad", title:"Author", "last-
➥name":"Dayley" };
  var obj2 = new Object();
  obj2.name = "Frodo";
  obj2.title = "Hobbit";
  obj2["last-name"] = "Baggins";
  document.write(obj1.name + " " + obj1["last-name"]
➥+ ", " + obj1.title + "<br>");
  document.write(obj2.name + " " + obj2["last-name"]
➥+ ", " + obj2.title);
</script>
```

ch0203.html

```
Brad Dayley, Author
Frodo Baggins, Hobbit
```

Output from ch0203.html

Defining Functions

```
function add(a, b){
  return a + b;
}
var result1 = add(5, 20);
```

You need to be familiar with creating and using functions in JavaScript. Functions are defined using the following syntax:

```
function function_name(arguments){function_block}
```

You can specify any number of arguments, and because JavaScript is loosely typed, you can pass in different object types each time you call the function. To call the function, simply reference it by name and pass in the desired arguments enclosed in `()` brackets. The code in the `function` block is then executed. You can use the `return` keyword to return a value from the function call.

The following code snippet shows examples of defining and calling functions:

```
<script>
  function add(a, b){
    return a + b;
  }
  var result1 = add(5, 20);
  var result2 = add("Java", "Script")
  document.write(result1 + "<br>");
  document.write(result2);
</script>
```

ch0204.html

```
25
JavaScript
```

Output from ch0204.html

Manipulating Strings

Strings are one of the most important data structures in JavaScript. Strings represent data that is conveyed to the user on the web page, whether it is a paragraph element, the name on a button, a menu label, or a figure caption. Strings are also used in the background to define locations, filenames, and a variety of other values used in web elements.

There are several special characters to define values that cannot be represented otherwise, such as new lines and tabs. Table 2.1 lists the special characters in JavaScript strings.

Table 2.1 **String Object Special Character**

Character	Description
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash
`\n`	New line
`\r`	Carriage return
`\t`	Tab
`\b`	Backspace
`\f`	Form feed

Getting the Length of a String

```
var s = "One Ring";
s.length; //returns 8
var s2 = "To Rule\nThem All";
s2.length; //returns 16 because \n is only 1
➥character
```

String objects have a length attribute that contains the number of characters in the string. This includes the number of special characters as well, such as `\t` and `\n`. To get the length of the string, simply use the `.length` attribute on the string object.

Finding What Character Is at a Specific Location in a String

```
var s = "In The Darkness Bind Them";
s.charAt(7); //returns 'D'
s[7]; // also returns 'D'
```

String objects provide the `charAt(offset)`, which returns the character contained at the offset specified. Also, strings in JavaScript are basically arrays of characters, so you can get the character at a specific offset using `stringName[offset]`. Keep in mind that the offsets are always zero based, so their first character is at offset 0.

Converting Numbers to Strings

```
var n=16;
var a = n.toString();  //a = 16
var b = n.toString(16); //b = 10
```

To convert a number to a string in JavaScript, create a variable with the numerical value if you don't already

have one and then call `.toString(radix)` on the number. The optional `radix` parameter specifies the base to use when converting the number. You can specify base `2` up to base `36`.

Converting Strings to Numbers

```
new Number("16");   //returns 16
new Number("0x20"); //returns 32
new Number("32.8"); //returns 32.8
```

To convert numerical-based strings into a number, create a new number object. JavaScript automatically detects whether the string is a number (even hex number with `0x##` formatting) and create a new number object. If the string is not a valid number format, a `NaN` object is returned.

Combining Strings

```
var str1="jQuery";
var str2=" & ";
var str3="JavaScript";
var str4 = str1.concat(str2, str3); //str5 = "jQuery
➥& JavaScript"
var str5 = str4 + " Phrasebook";    //str5 = "jQuery
➥& JavaScript Phrasebook"
```

You can combine multiple strings using the `.concat(str, str, …)` method. Or you can just use the `str + str + str . . .` method.

Changing String Case

```
var s1 = "jQuery and JavaScript";
var s2  = s1.toLowerCase();  //s2 = "jquery &
➥javascript"
var s3  = s1.toUpperCase(); //s3 = "JQUERY &
➥JAVASCRIPT"
```

Strings have built-in functions to change the case. The `.toLowerCase()` method returns a lowercase version of the string. The `.toUpperCase()` method returns an uppercase version of the string.

Splicing Strings

```
var s1 = "The play's the thing";
var s2 = s1.splice(4,8); // s2 = "play"
```

You can carve a string into substrings using `.splice(start, end)` by specifying the starting index and the ending index. The section of the string starting with the character at the index specified by `start` and ending with the character just before `end` index is returned. All indexes are zero based.

Splitting Strings

```
var s1 = "on-the-way-to-the/forum";
var arr = s1.split("-"); // arr = ["on", "the",
➥"way", "to", "the","forum"];
```

To split a string into chunks using a specific delimiter, use the `.split(separator [, limit])`. The separator defines where to split the string and is not included in the results. The limit specifies the number of items to return before stopping. An array of the split portions of the string is returned.

Checking to See If a String Contains a Substring

```
var s1 = "I think therefore I am";
var a = s1.indexOf("think");     // a = 2
var b = s1.indexOf("thought"); // b = -1
```

The simplest way to check to see if one string is a substring of another is to use `.indexOf(substring)` on the main string. This function returns an index of the location of the first occurrence, or –1 if the substring is not found.

Finding and Replacing Substrings Regardless of Case

```
var s1 = "jQuery, sometimes JQuery or JQUERY";
var s2 = s1.replace(/jQuery/gi, "jQuery");
//s2 is now "jQuery, sometimes jQuery or jQuery"
```

The best way to find and replace substrings while ignoring case is to use a regular expression in the `.replace()` method. If you are not familiar with them, I'd suggest you at least look into them. They can use complex formulas for finding matches in strings. For now, I'll just show you a simple expression.

As an example, to find all instances of the term `jQuery` when you may not know the case specified in the string, you would define the following regular expression. The `g` character specifies a global search (meaning not to stop at the first instance). The `i` character specifies to ignore case.

```
/jQuery/gi
```

Manipulating Arrays

One of the most useful data structures in JavaScript is arrays. Arrays are collections of items that can be of any type. You have already seen how to create and access the individual items. This section contains phrases designed to show you how to manipulate the

arrays by combining them, slicing them, searching for items, and more.

Combining Arrays

```
var fruits = ["banana", "apple", "orange"];
var veggies = ["broccoli", "carrots", "spinach"];
var grains = ["wheat", "oats"];
var food = grains.concat(fruits, veggies);
//food = ["wheat","oats","broccoli","carrots",
➥"spinach","banana","apple","orange"]
```

You can combine multiple arrays using the `.concat(arr, arr, …)` method. This returns a new array object with the array elements combined inside.

Splicing Arrays

```
var week = ["sun", "Mon", "Tue", "Wed", "thur",
➥"Fri", "sat"];
var weekdays = week.slice(1,6);
```

You can carve arrays the same way you can strings by using the `.splice(start, end)` method and specifying the starting index and the ending index. The items in the array beginning with the start index until the item just before the end index are returned in a new array object. All indexes are zero based.

Detecting Whether an Array Contains an Item

```
var food = ["broccoli", "carrots", "spinach"];
var a = food.indexOf("spinach");     // a = 2
var b = food.indexOf("pizza"); // b = -1
```

The simplest way to check whether an item already exists in an array is to use the `.indexOf(item)`. If the item is found, the index of the first instance is returned; otherwise, –1 is returned.

Removing Items from the Array

```
var week = ["sun", "Mon", "Tue", "Wed", "thur",
➥"Fri", "sat"];
var weekdays = week.splice(1,5);
var day = week.pop();
// weekdays= ["Mon", "Tue", "Wed", "thur", "Fri"]
// day = "sat"
// week = ["sun", "Mon", "Tue", "Wed", "thur",
➥"Fri"];
```

You can remove the last item in the array using `pop()`. If you want to remove items in the middle of the array, you need to use `splice(index, count)`, where `index` is the index to begin removing items from and `count` is the number of items to remove.

Creating a Tab-Separated String from an Array

```
var weekdays= ["Mon", "Tue", "Wed", "thur", "Fri"]
var weekStr = weekdays.join("\t");
// weekStr = "Mon\tTue\tWed\tthur\tFri"
```

The `.join(separator)` method allows you to combine an array into a string. The optional separator argument specifies the character or string to place between each item in the newly created string.

Sorting Arrays

```
arr1.sort(function(a,b){return a-b;});
...
arr1.sort(function(a,b){return Math.abs(a)-
➥Math.abs(b);});
...
arr1.sort();
```

JavaScript provides a nice interface to sort arrays. The array object has a `sort(function)` method. The `sort()` method sorts the elements of the array in alphabetical order, converting elements to strings if possible, which is a problem for numerical arrays. You can specify your own sort function that accepts two elements and returns the following:

- **true** or a **positive number** if the first value should be sorted higher
- **0** if they are equal
- **false** or a **negative number** if the first value should be sorted lower

```
<script>
  var arr1 = [-10, -5, -1, 0, 2, 6, 8];
  var arr2 = ["a","b", "c", "A", "B", "C"];
  arr1.sort(function(a,b){return a-b;});
  document.write(arr1 + "<br>");
  arr1.sort(function(a,b){return Math.abs(a)-
➥Math.abs(b);});
  document.write(arr1 + "<br>");
  arr1.sort();
  document.write(arr1 + "<br>");
  arr2.sort();
  document.write(arr2 + "<br>");
  arr2.sort(function(a,b){return
➥a.toUpperCase()>b.toUpperCase();});
```

```
  document.write(arr2 + "<br>");
</script>
```

ch0205.html

```
-10,-5,-1,0,2,6,8
0,-1,2,-5,6,8,-10
-1,-10,-5,0,2,6,8
A,B,C,a,b,c
A,a,B,b,C,c
```

Output from ch0205.html

Applying Logic

Like any other programming language, JavaScript provides the basic logic operators and operations. The following sections provide phrases on using the JavaScript comparison operators to generate `if/else` blocks, loops, and a variety of other logic.

Determining If an Object Is Equal To or Less Than Another

```
var x=5;
var y="5";
var resultA = x==y; //Evaluates to true
var result = x===y; //Evaluates to false
```

JavaScript uses the standard `==, !=, >, <, >=, <=` operators to compare two objects. It also includes the `===` and `!==` operators that are type sensitive. For example, `1=="1"` evaluates to `true`, whereas `1==="1"` does not because the first is a numerical type and the second is a string type.

Adding Conditional Blocks of Code

```
if (age >= 100){
  document.write("Wow you are a centurian!");
} else if (age >= 18) {
  document.write("You are an adult.");
} else {
  document.write("You are a child.");
}
```

If blocks are created using the basic `if (comparison) {IF_BLOCK}`. The `if` block can be followed up by an additional `else` block using `else{ELSE_BLOCK}`. In addition, you can add a condition to the `else` block using `else(condition){ELSE_BLOCK}`.

Using a While Loop

```
var weekdays = ["Mon", "Tue", "Wed", "Thur", "Fri"];
do{
  document.write(day);
  var day = weekdays.pop();
  document.write(day+", ");
} while (day != "Wed");
//Output is "Fri, Thur, Wed, "
```

The `while(condition){LOOP_BLOCK}` and `do{LOOP_BLOCK}while(condition);` loops allow you to loop through code until a specific condition occurs. The code in your `LOOP_BLOCK` is executed each time through the loop.

Iterating Through Arrays

```
var weekdays = ["Mon", "Tue", "Wed", "Thur", "Fri"];
for(var x=0; x < weekdays.length; x++){
  document.write(day+"|");
}
//Output is "Mon|Tue|Wed|Thur|Fri|"
```

The best method to iterate through JavaScript arrays is to use a `for(init; condition; adjustment){LOOP_BLOCK}` loop. The `for()` loop allows you to initialize a variable and then iterate through until a condition is met. To loop through the array, initialize an index and then stop the loop when the index is equal to the length of the array.

Iterating Through Object Properties

```
var obj = {first:"Bilbo", last:"Baggins",
➥title:"Hobbit"};
for (var key in obj){
  document.write(key + "=" + obj[key] + "&");
}
```

JavaScript provides an additional type of option in the `for()` loop. Using the `in` keyword, you can iterate through the values in an array or the properties in an object. The syntax is `for( var name in object){LOOP_BLOCK}`. The name is assigned the property name inside the `LOOP_BLOCK`.

Math Operations

JavaScript has a built-in `Math` object that provides functionality for extended mathematical operations. The `Math` object has property values, listed in Table 2.2, that contain the most common constants. You can access the `Math` object directly from your JavaScript code. For example, to get the value of pi, you would use the following:

```
Math.PI
```

Table 2.2 **Math Object Properties**

Property	Description
`E`	Returns Euler's number (approx. 2.718)
`LN2`	Returns the natural logarithm of 2 (approx. 0.693)
`LN10`	Returns the natural logarithm of 10 (approx. 2.302)
`LOG2E`	Returns the base-2 logarithm of E (approx. 1.442)
`LOG10E`	Returns the base-10 logarithm of E (approx. 0.434)
`PI`	Returns PI (approx. 3.14)
`SQRT1_2`	Returns the square root of 1/2 (approx. 0.707)
`SQRT2`	Returns the square root of 2 (approx. 1.414)

The `Math` object also includes several methods that allow you to apply higher math operations. The following sections provide phrases for those operations.

Generating Random Numbers

```
// Random number between 0 and 1
Math.random();
//Random number between 1 and 10
Math.floor(Math.random() * 10) + 1;
//Random number between -10 and 10
Math.floor(Math.random() * 20) - 10;
```

The `Math.random()` function generates a random float value between 0 and 1. To convert that to an integer, multiply the results of `Math.random()` by the maximum value you want for the integer and then use `Math.floor()` to round it down.

Rounding Numbers

```
var x = 3.4;
var y = 3.5;
Math.round(x); //returns 3
Math.round(y); //returns 4
Math.ceil(x); //returns 4
Math.ceil(y); //returns 4
Math.floor(x); //returns 3
Math.floor(y); //returns 3
```

The JavaScript `Math` object provides three methods to round numbers. The `.round()` method rounds numbers `#.5` up to the next higher integer and numbers `#.49999…` down to the next lowest integer. The `.ceil()` always rounds up to the next highest integer. The `.floor()` always rounds down to the next lowest integer.

Getting the Minimum Number in a Set

```
Math.min(8,"0x5",12,"44",8,23,77); //returns 5
var arr1 = [4,8,12,3,7,11];
Math.min.apply( Math, arr1 ); //returns 3
var arr2 = ["0x5",8,12,"4",8,23,77,"0x1F"];
Math.min.apply( Math, arr2 ); //returns 4
```

The `Math.min(item, item, …)` method allows you to find the smallest number in a set. You can also apply an array object to the `.min()` method using `Math.min.apply(Math, array)`. This returns the smallest number in the array. The array can contain string representations of numbers, such as `"4"` and `"0x5B"`, but not character strings such as `"A"`. If the array contains character strings, the result will be `NaN`.

Getting the Maximum Number in a Set

```
Math.max(8,"0x5",12,"44",8,23,77); //returns 77
var arr1 = [4,8,12,3,7,11];
Math.max.apply( Math, arr1 ); //returns 12
var arr2 = ["0x5",8,12,"4",308,23,77,"0xFF"];
Math.max.apply( Math, arr2 ); //returns 308
```

The `Math.max(item, item, …)` method allows you to find the largest number in a set. You can also apply an array object to the `.max()` method using `Math.max.apply(Math, array)`. This returns the largest number in the array. The array can contain string representations of numbers such as `"26"` and `"0x1F"`, but not character strings such as `"A"`. If the array contains character strings, the result will be `NaN`.

Calculating Absolute Value

```
Math.abs(-5); //returns 5
Math.abs(5); //returns 5
Math.abs("-15"); //returns 15
Math.abs("0xFF"); //returns 255
```

The `Math.abs(x)` method returns the absolute value of `x`. The variable `x` can be a number or a numerical string such as `"5"`, `"-5"`, or `"0xFF"`.

Applying Trigonometric Functions

```
Math.log(2); //returns 0.693
Math.sin(0); //returns 0
Math.cos(0); //returns 1
Math.tan(1); //returns 1.5574
```

The Math object provides several trigonometric methods that apply trig functions to numbers. Table 2.3 lists the trig methods to apply things such as natural logs, sine, cosine, and tangent operations.

Table 2.3 **Match Object Trigonometric Methods**

Method	Description
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a number between -PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
cos(x)	Returns the cosine of x (x is in radians)
log(x)	Returns the natural logarithm (base E) of x
sin(x)	Returns the sine of x (x is in radians)
tan(x)	Returns the tangent of an angle

Applying Power Functions

```
Math.exp(4); //returns 54.59815
Math.pow(2,16); //returns 65536
Math.sqrt(25); //returns 5
```

The Math object provides several power functions that allow you to apply exponential and root operations on numbers. Table 2.4 lists the trig methods to apply Euler's exponential, power, and square root functions. You can pass in the values as numbers or numerical strings.

Table 2.4 **Match Object Trigonometric Methods**

Method	Description
`exp(x)`	Returns the value of E^x
`pow(x,y)`	Returns the value of x to the power of y
`sqrt(x)`	Returns the square root of x

Working with Dates

A useful built-in JavaScript object is the `Date` object. The `Date` object allows you to get the user's time by accessing the system clock through the browser. This allows you to use JavaScript to get the local time for the user. The `Date` object also provides methods for manipulating the time values to determine time deltas and generate time and date strings.

Creating a Date Object

```
var d = new Date();
//Wed Jan 30 2013 09:37:42 GMT-0700
var d = new Date(1209516513246);
//Tue Apr 29 2008 18:48:33 GMT-0600
var d = new Date("May 17 2014 13:55:13 GMT-0600");
//Sat May 17 2014 13:55:13 GMT-0600
var d = new Date("1/1/2020");
//Sat Jan 01 2020 00:00:00 GMT-0700
var d = new Date("12/12/2012 12:12:12");
//Wed Dec 12 2012 12:12:12 GMT-0700
var d = new Date(2014, 8, 6, 16, 5, 10, 3);
//Sat Sep 06 2014 16:05:10 GMT-0600
```

A `Date` object represents a specific date and time with millisecond granularity. To create a `Date` object, all you need to do is call a new `Date()` using one of the following methods. The `dateString` can be in one of several different formats, such as `"7-17-2013"`, `"11/22/2014`

10:12:05", "Dec 12 2012 12:12:12", and "2014-01-05T08:03:22.356Z".

- new Date();
- new Date(milliseconds);
- new Date(dateString);
- new Date(year, month, day, hours, minutes, seconds, milliseconds);

Getting the Current Date and Time

```
new Date();
```

To get the current date and time, create a new Date() object without passing parameters. This creates an object with the current date and time.

Creating a Date String

```
var d = new Date();
d.toDateString();
//Returns Sat May 17 2014
d.toLocaleDateString();
//Returns Saturday, May 17, 2014
d.toISOString();
//Returns 2014-05-17T19:55:13.000Z
d.toUTCString();
//Returns Sat, 17 May 2014 19:55:13 GMT
d.toString();
//Returns Sat May 17 2014 13:55:13 GMT-0600
➥(Mountain Daylight Time)
```

There are several methods to generate a date string from JavaScript. The first step is to get a Date object. The Date object provides the following methods that return differently formatted date strings:

- `.toDateString()`—Returns an abbreviated date string.
- `.toLocaleDateString()`—Returns a localized date string.
- `.toISOString()`—Returns the ISO standardized date string.
- `.toUTCString()`—Returns a date string converted to UTC time.
- `.toString()`—Returns the full date string with time zone and more.

Creating a Time String

```
var d = new Date();
d.toLocaleTimeString()
1:55:13 PM
d.toTimeString()
13:55:13 GMT-0600 (Mountain Daylight Time)
```

The `Date` object also allows you to get just a time string. There are two methods attached to the `Date` object that return a time string. The `.toTimeString()` method returns a time string with the time, time zone, and even daylight savings time information. The `.toLocaleTimeString()` returns a more basic time string with local formatted time.

Getting a Time Delta

```
var atime = new Date("2014-01-05T08:03:22.356Z");
var btime = new Date("2014-01-05T09:08:57.758Z");
var delta = Math.abs(btime - atime); //3935402ms
var totalSec = Math.floor(delta * .001); //3935s
var hours = Math.floor(totalSec / 3600); //1hr
var minutes = Math.floor(totalSec / 60) % 60; //5min
var seconds = totalSec % 3600 % 60; //35sec
```

`Date` objects essentially represent the number of milliseconds from Jan 1, 1970. Therefore, you can get the time difference between two `Date` objects by subtracting them from each other. The result is a number that represents the millisecond difference between the two dates.

By the way

When subtracting two dates to get the millisecond delta between them, you can end up with a negative number depending on which date is subtracted from the other. Use `Math.abs(timeA-timeB)` to always generate a positive time delta.

Getting Specific Date Components

```
var d = new Date("Sat Jan 04 2014 15:03:22 GMT-
➥0700");
d.getDate(); //returns 4
d.getDay(); //returns 6
d.getFullYear(); //returns 2014
d.getHours(); //returns 15
d.getMinutes(); //returns 3
d.getMinutes(); //returns 22
d.getTime(); //returns 1388873002000
d.getTimeZoneOffset(); //returns 420
```

The `Date` object provides several methods to get the value of specific components of the date, such as year, day of month, day of week, and more. Table 2.5 lists the methods to get specific date components from the `Date` object.

Table 2.5 **Date Object Methods to Get Specific Date Components**

Method	Description
`getDate()`	Day of the month (1–31)
`getDay()`	Day of the week (0–6)
`getFullYear()`	Year (4 digits)
`getHours()`	Hour (0–23)
`getMilliseconds()`	Milliseconds (0–999)
`getMinutes()`	Minutes (0–59)
`getMonth()`	Month (0–11)
`getSeconds()`	Seconds (0–59)
`getTime()`	Milliseconds since 1-1-1970 12:00:00
`getTimezoneOffset()`	Difference between UTC time and local time, in minutes

Did you know?

Date objects also have methods to obtain the UTC components by placing UTC after get, such as `getUTCHour()` and `getUTCSeconds()`.

```
<script>
  var atime = new Date("2014-01-04T08:45:22.356Z");
  var btime = new Date("2014-01-04T12:12:57.758Z");
  var delta = Math.abs(btime - atime); //3935402ms
  var totalSec = Math.floor(delta * .001); //3935s
  var hours = Math.floor(totalSec / 3600); //1hr
  var minutes = Math.floor(totalSec / 60) % 60;
➥//5min
  var seconds = totalSec % 3600 % 60; //35sec
```

```
  document.write("Total Milliseconds: " + delta +
➥"<br>");
  document.write("Total Seconds: " + totalSec +
➥"<br>");
  document.write("Delta: " + hours + "hrs ");
  document.write(minutes + "secs " + seconds + "s");
</script>
```

ch0206.html

```
Total Milliseconds: 12455402
Total Seconds: 12455
Delta: 3hrs 27secs 35s
```

Output from ch0206.html

This page intentionally left blank

3

Interacting with the Browser

Dynamic web pages often require you to access and in some cases even manipulate things beyond the HTML elements. JavaScript provides a rich set of objects and functions that allow you to access information about the screen, browser window, history, and more.

The phrases in this chapter describe ways to use the `screen`, `window`, `location`, and `history` objects that provide JavaScript with an interface to access information beyond the web page. Additional phrases describe utilizing those objects to implement cookies, popup windows, and timers.

Writing to the JavaScript Console

```
console.log("This is Debug"); //"This is Debug" is
➥displayed in console
var x=5;
console.log("x="+x): //"x=5" is displayed in console
```

The JavaScript console can be an important tool when debugging problems in your jQuery and JavaScript code. The console log is simply a location where you can view data output from the JavaScript code. Each of the major browsers has a console log tool that displays the output.

To output data to the console log, use `console.write(DEBUG_STRING)` and pass it the text that you want to display on the console.

Reloading the Web Page

```
location.reload()
```

A useful feature of JavaScript is the ability to force the browser to reload the current web page. You may want to reload the web page because data has changed on the server or a certain amount of time has elapsed. The `location.reload()` method requests that the browser reload the current URL.

Redirecting the Web Page

```
location.href="newpage.html";
location.href="http://jqueryin24/index.html;
```

Another extremely useful feature of JavaScript is the ability to redirect the browser away from the current URL to another location. You do this by setting the `location.href` value to a new URL. The new URL can be a full address, such as http://mysite.com/newlocation/index.html, or a relative location to the current URL, such as page2.html.

Getting the Screen Size

```
screen.availHeight; // returns screen height in
➥pixels
screen.availWidth; // returns screen width in pixels
```

An important feature of JavaScript these days is the ability to get the screen size. The screen sizes of browsers vary so much that you often need to use different sets of code for larger, medium, or smaller screens. To get the screen size, use the `screen.availHeight` and `screen.availWidth` attributes. These values are specified in the number of pixels.

Getting Current Location Details

The JavaScript location object provides an easy way to get various pieces of information about the URL the user is currently viewing. Because JavaScript and jQuery code are often used across multiple pages, the location object is the means to get information about what URL the browser is currently viewing. The following sections provide some phrases that provide information about the browser's current location.

Finding the Current Hash

```
location.hash
```

The `location.hash` value returns the current hash, if any, that the browser is using. This is useful when you are displaying a web page with multiple hash anchors. The hash provides a context to the location on the page the user actually clicked.

Getting the Host Name

```
location.hostname
```

The `location.hostname` value returns the domain name of the server that sent the page to the user. This is just the portion after the protocol but before the port number or path. This allows you to determine which server to send AJAX requests back to. Often, multiple servers may handle the web site, but you may want to interact only with the server that served the web page in the first place.

Looking at the File Path

```
location.pathname
```

The `location.pathname` returns the path that the page was loaded from on the server. The pathname also provides a context as to what page the user is looking at.

Getting the Query String

```
location.search
```

The `location.search` value returns the query string that was passed to the server to load the page. Typically, you think about a query string as a server-side script tool. However, it can be just as valuable to JavaScript code that manipulates the data returned from the server and requests additional information from the server via AJAX.

Determining If the Page Is Viewed on a Secure Protocol

```
location.protocol
```

The easiest way to determine if a page is being viewed from a secured location on the server is to look at the `location.protocol` value. This value will be http on regular requests or https on secure requests.

Accessing the Browser

Another important object built into JavaScript is the window object. The window object represents the browser and provides you with a wealth of information about the browser position, size, and much more. It also allows you to open new child windows, close windows, and even resize the window.

Getting the Max Viewable Web Page Size

```
window.innerHeight; // returns browser view port
➥height in pixels
window.innerWidth; // returns browser view port
➥width in pixels
```

The window object provides the `innerHeight` and `innerWidth` of the browser window. These values represent the actual pixels in the browser window that the web page will be displayed within. This is a critical piece of information if you need to adjust the size and location of elements on the web page based on the actual area that is being displayed.

Setting the Text Displayed in the Browser Status Bar

```
window.status = "Requesting Data From Server . . .";
```

The browser has a status bar at the bottom. You can set the text that is displayed there to provide to the user additional information that does not belong on the page, such as the server name, current status of requests, and more. To set the text displayed in the browser status bar, set the `window.status` value equal to the string you want displayed.

Getting the Current Location in the Web Page

```
window.pageXOffset;
// number of pixels the page has scrolled to the
➥right
window.pageYOffset;
// returns number of pixels the page has scrolled
➥down
```

When writing dynamic code, it is often necessary to determine the exact location in the web page that is currently being viewed. When the user scrolls down or to the right, the position of the page to the frame of the browser view port changes.

To determine the number of pixels the page has scrolled to the right, use the `window.pageXOffset` attribute. To determine the number of pixels the page has scrolled down, use the `window.pageYOffset` attribute.

Opening and Closing Windows

```
//Opens a new blank window, writes to it, and then
➥closes it.
var newWindow = window.open();
newWindow.document.write("Hello From a New Window");
newWindow.close();
//Opens another URL in a new window
window.open("http://google.com");
```

The window object also provides a set of methods that allow you to create and manage additional child windows from your JavaScript code.

For example, the `window.open(URL)` method opens a new window and returns a new window object. If you do not specify a URL, the browser opens a blank page that can be written to using the `window.document` object.

You can call `.close()` on window objects that you have created, and they will be closed.

Using the Browser History to Go Forward and Backward Pages

```
history.forward(); //forward 1 page
history.back(); //backward 1 page
history.go(-2); //backward 2 pages
```

The browser keeps track of the pages that have been navigated to in a history. JavaScript allows you to access this history to go forward or backward pages. This allows you to provide forward and backward controls to your web pages. You can also use this feature to provide bread crumbs displaying links to multiple pages back in the history.

To go forward one page, use `history.forward()`. To go backward one page, use `history.back()`.

To go forward or backward multiple pages, use `history.go(n)`, where `n` is the number of pages. A negative number goes backward that many pages, and a positive number goes forward that many pages.

Creating Popup Windows

```
var result = confirm("You Entered " + response +
  "is that OK?");
if(result){ alert("You Said Yes.") }
else {alert("You Said no.")}
```

Window objects provides several different methods that allow you to launch popup windows that you can interact with for alerts, prompts, and notifications. The popup windows are displayed, and the user needs to interact with the popup before continuing to access the web page.

There are three kinds of popup boxes that can be created:

- **`alert(msg)`**—Launches a popup window that displays an alert message and provides a button to close the popup.
- **`confirm(msg)`**—Launches a popup window that displays a confirmation and message provides an OK and a Cancel button, which both close the popup. If you click the OK button, the return value from `confirm()` is true; otherwise, it is false.

- **`prompt(msg)`**—Launches a popup window that displays the message, a text box for input, and an OK and Cancel button, which both close the popup. If you click the OK button, the return value from `prompt()` is the text typed into the text box; otherwise, it is false.

The code that follows illustrates these popup boxes, as shown in Figure 3.1.

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     var response = prompt("What is the airspeed "
➥+
09     "velocity of an unladen swallow:");
10     var result = confirm("You Entered " +
➥response +
11       "is that OK?");
12     if(result){ alert("You may pass.") }
13     else {alert("None Shall Pass.")}
14   </script>
15 </head>
16 <body>
17 </body>
18 </html>
```

ch0301.html

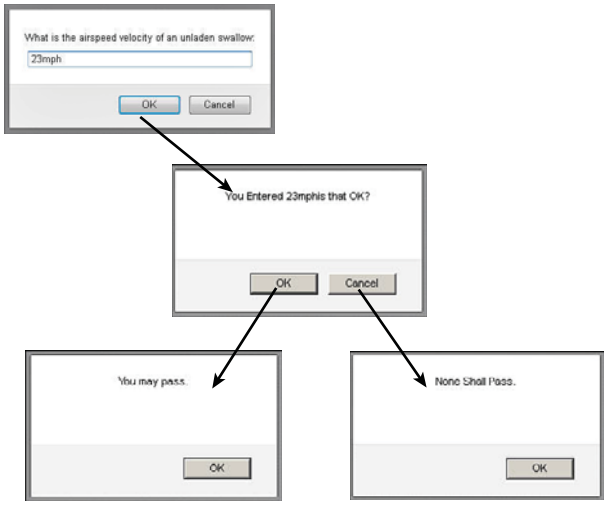


Figure 3.1 Various popup boxes, with the results being passed from popup to popup using JavaScript.

By the way

It is often much better to create a fixed position `<div>` element with an overlay than to use these popup boxes because you have much more control over them. I'll show you how to do just that a little later in the book.

Manipulating Cookies

A common task in JavaScript is getting and setting cookies in the browser. Cookies allow you to store simple key value pairs of information in the browser in a persistent manner. You can access the cookies by the

server-side scripts and JavaScript to determine what those values are.

You can access cookie information using the `document.cookie` object. This object contains all the cookies in the string format `name=value; name=value; ....`

Setting a Cookie Value in the Browser

```
function setCookie(name, value, days) {
  var date = new Date();
  date.setTime(date.getTime()+(days*24*60*60*1000));
  var expires = "; expires="+date.toGMTString();
  document.cookie = name + "=" + value +
                    expires + "; path=/";
}
```

To add a new cookie for your web site, set `document.cookie = "name=value; expireDate; path;";`. The expire date needs to be a date set using `.toGMTString()`, and the path is the path on your web site that the cookie applies to.

Getting a Cookie Value from the Browser

```
function getCookie(name) {
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i].split("=",2);
    cookie[0] = cookie[0].replace(/^\s+/,"");
    if (cookie[0] == name){ return cookie; }
  }
}
```

To get the value of the cookie, split the `document.cookie` value using the `;` character, and then iterate

through the resulting array until you find the name you are looking for.

Example Getting and Setting Cookies

The following code shows a full example of setting and getting cookies. When the code is run, two cookies are set: one for name and the other for language. The cookies are then retrieved from the browser and written to the web page, as shown in Figure 3.2.

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06   src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     function setCookie(name, value, days) {
09       var date = new Date();
10       date.setTime(date.getTime()+(days*24*60*60*
➥1000));
11       var expires = "; expires="+date.toGMTString
➥();
12       document.cookie = name + "=" + value +
13                         expires + "; path=/";
14     }
15     function getCookie(name) {
16       var cArr = document.cookie.split(';');
17       for(var i=0;i < cArr.length;i++) {
18         var cookie = cArr[i].split("=",2);
19         cookie[0] = cookie[0].replace(/^\s+/,"");
20         if (cookie[0] == name){ return cookie; }
21       }
22    }
```

```
23     setCookie("name", "Brad", 1);
24     setCookie("language", "English", 1);
25     document.write("<h3>Cookies</h3>");
26     var c1 = getCookie("name");
27     document.write(c1[0] + " is set to "+ c1[1]
➥+"<br>");
28     var c2 = getCookie("language");
29     document.write(c2[0] + " is set to " +
➥c2[1]);
30   </script>
31 </head>
32 <body>
33 </body>
34 </html>
```

ch0302.html

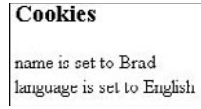


Figure 3.2 Adding cookie output using JavaScript.

Adding Timers

Another useful feature of JavaScript is the ability to set timers that execute a function or evaluate an expression after a certain amount of time or on a specific interval.

Using timers allows you to delay the execution of code so that it does not need to happen at the exact moment an event is triggered or the page is loaded. The following phrases show you how to create timers to delay code execution and to apply recurring actions.

Adding a Delay Timer

```
function myTimer () {
  alert("Timer Function Executed");
}
var timerId = setTimeout(myTimer, 10000);
```

To simply delay the execution of code for a certain amount of time, use the `setTimeout(code, ms)` method, where `code` is either a statement or a function that executes when the time expires. `ms` is the number of milliseconds. For example, to execute a function named `myTimer()` in 10 seconds, you would use the following:

```
setTimeout(myTimer, 10000);
```

Cancel a Timer

```
function myTimer () {
  alert("Timer Function Executed");
}
var timerId = setTimeout(myTimer, 10000);
clearTimeout(timerId); //timer will not execute
```

At any point before the time runs out and the code is executed, you can clear the timer by calling the `clearTimeout(id)` method using the ID returned from `setTimeout()`. Once the timer has been cleared, it does not execute. You need to keep track of the ID returned from `setTimeout()` using a variable.

Adding a Recurring Timer

```
var statusTimerId;
var status = "OK";
//checks status every minute as long as it is "OK"
function checkStatus () {
  if(status == "OK"){
```

```
    alert("Status OK");
    statusTimerId = setInterval(checkStatus, 60000);
  } else {
    alert("Status Failed");
  }
}
statusTimerId = setInterval(checkStatus, 60000);
```

You can also start a timer that triggers on a regular interval using the `setInterval(function, ms)` method. This method also accepts a function name and milliseconds as arguments. Inside the function, you need to call set interval again on the same function so that the code will be called again.

To turn off the recurring timer, simply do not call `setInterval()` again inside the timer function, or use `clearInterval()` outside the function.

This page intentionally left blank

4

Accessing HTML Elements

The most important part of dynamic web development is the ability to access the DOM elements quickly and efficiently. JavaScript inherently provides functionality to access the DOM elements. This JavaScript feature can be useful at times, but this is the area where jQuery really stands out. At times you will need to use JavaScript methods to access the DOM elements, but when possible I recommend using jQuery.

The phrases in this chapter cover both the JavaScript and jQuery selector methods to find DOM elements.

Finding HTML Elements in JavaScript

There are three ways to find HTML elements using JavaScript. You can search for them by the ID attribute name, by the class name, or by the tag type. The following code shows an example of defining an HTML `<div>` element so that each of these methods can use it:

```
<div id="myDiv" class="myClass">Content</div>
```

By adding the ID and class attributes, you can use those values of "myDiv" and "myClass" to search for the element.

Finding DOM Objects by ID

```
var containerObj = document.getElementById("myDiv");
```

The simplest method to find an HTML element is to use the value of the id attribute with the document.getElementById(id) function. The document.getElementById(id) function searches the DOM for an object with a matching id attribute. If that object is found, the function returns the DOM object.

Finding DOM Objects by Class Name

```
var objs =
➥document.getElementsByClassName("myClass");
for (var i=0; i<objs.length; i++){
  var htmlElement = objs[i];
  ...
}
```

You can also search for HTML elements by their class attribute using the document.getElementsByClassName(class). This function returns an array of DOM objects with matching class attributes. You can then iterate over that list using a JavaScript loop and apply changes to each DOM element.

Finding DOM Objects by Tag Name

```
var objs = document.getElementsByTagName("div");
for (var i=0; i<objs.length; i++){
  var htmlElement = objs[i];
  ...
}
```

Another way search for HTML elements is by their HTML tag using the `document.getElementsByTagName(tag)`. This function searches the document for all DOM objects that have the specified tag name and returns them in an array. You can then iterate over that array using a JavaScript loop and access or apply changes to each DOM element.

Using the jQuery Selector to Find HTML Elements

Unlike JavaScript, jQuery allows you to find HTML elements in countless ways using selectors. Yes, just like CSS selectors. In fact, most jQuery selectors are based on the CSS selectors, thus providing a more seamless transition between the two.

As demonstrated by the phrases in the upcoming sections, jQuery selectors make it easy to select just about any group of HTML elements. Keep in mind that jQuery selectors return jQuery objects that represent a set of DOM objects, not a direct array of DOM objects.

jQuery selector syntax is straightforward. Once the jQuery library is loaded, simply use `$(selector)`. For example:

```
$("#myElement")
```

Watch out!

There are several meta characters used in jQuery selector syntax. If you want to use any of the meta characters, such as `!"#$%&'()*+,./:;<=>?@[\]^`{|}~`, as a part of a class/ID/name, you need to escape the character with `\\` (two backslashes). For example, if you have an element with `id="my.element"`, you would use the selector `$("#my\\.element")`.

Applying Basic jQuery Selectors

```
$("*"); //selects all elements
$(".myClass"); //selects elements with
➥class="myClass"
$("#myDiv"); //selects the element with id="myDiv"
$("div"); //selects <div> elements
$("div, span, p"); //selects <div>, <span>, and <p>
➥elements
$("div.myClass"); //selects <div> elements with
➥class="myClass"
```

The most commonly used selectors are the basic ones. The basic selectors focus on the ID attribute, class attribute, and tag name of HTML elements. Table 4.1 lists some examples to show you how to define some of the basic selectors.

Table 4.1 **Examples of Using Basic jQuery Selectors**

Syntax/Example	Description
`$("*")`	Selects all HTML elements.
`$(".class")` `$(".container")`	Selects elements based on the `class` attribute. Example: Selects all HTML elements with `class="container"`. The `.` character prefix denotes a class name.

Syntax/Example	Description
`$("#id")` `$("#menu")`	Selects an element based on the `id` attribute. Example: Selects the HTML elements with `id="menu"`. The `#` character prefix denotes an `id` value.
`$("element")` `$("div")`	Selects elements based on tag type. Example: Selects all the `<div>` elements.
`$("element,element...")` `$("div, span, p")`	Selects multiple types of elements based on tag. Example: Selects all `<div>`, `<span>`, and `<p>` elements. You can also specify multiple elements separated by a comma.
`$("element.class")` `$("ul.bigLists")`	Selects elements of a specific tag and class. Example: Selects all `<ul>` elements that have `class="bigList"` by combining the element name and class. Note that there is no space between the element and the class name.

Selecting Elements Based on HTML Attributes

```
$("input[value=0]"); //selects <input> with
➥value="0"
$("p[class*=my")//selects elements "my" in classname
$("img[src^='icons\\/']");
  //selects <img> elements where src starts with
➥"icons/"
$("input[value!='default']");
  //selects <input> elements where the value is not
➥"default"
$("p[id]"); //selects <p> elements that have id set
$("p[id][class$=Menu");
  //selects <p> elements with idset and
  //classname ending with "Menu"
```

Another way to use jQuery selectors is to select HTML elements by their attribute values. It can be a default attribute or one that you have added to the HTML elements. Attribute values are denoted in the selector syntax by being enclosed in `[]` brackets. Table 4.2 shows some of the ways that you can apply attribute selectors.

Table 4.2 **Examples of Using Attribute jQuery Selectors**

Syntax/Example	Description
`$([attribute=value]")` `$("input[value=0]")`	Selects elements where attribute `attr=value`. Example: Selects all `<input>` elements that have a `value` attribute equal to 0.

Syntax/Example	Description
`$([attr*=value])`	Attribute `attr` contains `value`.
`$("p[class*=Content")`	Example: Selects all HTML elements with `"Content"` in the class name. For example, all of the following `<p>` elements would be selected: `<p class="leftContent">...</p>` `<p class="centerContent">...</p>` `<p class="rightContent">...</p>`
`$("[attr^=value]")`	Attribute `attr` begins with value.
`$("img[src^='icons\\/']")`	Example: Selects all `<img>` elements whose `src` attribute starts with `"icons/"`. Notice that because the expression was not simple text, quotes were required around the value. Also notice that the / character had to be escaped with \\.
`$("[attr!=value]")`	Attribute `attr` does not equal value.
`$("input[value!='default text']")`	Example: Selects all the `<input>` elements where the value does not equal `"default text"` or they do not have a `value` attribute.
`$("[attr]")`	Selects elements that have attribute `attr`.
`$("p[id]")`	Example: Selects all `<p>` elements that have an `id` attribute.
`$("[attr][attr2$=value])`	Selects elements that have attribute `attr` and have attribute `attr2` equal to `value`.

Table 4.2 **Continued**

Syntax/Example	Description

`$("p[id] ➥[class$=Menu")`	Example: Selects all `<p>` elements that have an `id` attribute and have a `class` attribute that ends with `"Menu"`. For example, only the top two of the following HTML elements would be selected:
	`<p id="topMenu" class="topMenu ➥">...</p>`
	`<p id="topMenu" ➥class="leftMenu">...</p>`
	`<p id="contentMenu" ➥class="contentMenuItem">...</p>`
	`<p class="contentMenu">...</p>`

Selecting Elements Based on Content

```
$("p:contains('free')");
  //selects <p> elements that contain "free
$("div:has(span)");
  //selects <div> elements that contain <span>
➥elements
$("div:empty");
  //selects <div> elements with no content or
➥children
$("div:parent");
  //selects <div> elements that have at least some
➥content
```

Another set of useful jQuery selectors are the content filter selectors. These selectors allow you to select HTML elements based on the content inside the HTML element. Table 4.3 shows examples of using content selectors.

Table 4.3 **Examples of Using Content jQuery Selectors**

Syntax/Example	Description
`$(":contains(value)")` `$("div:contains ('Open Source')")`	Selects elements that have the text in value in their contents. Example: Selects all `<div>` elements that contain the text `"Open Source"`.

`$(":has(element)")`
`$("div:has(span)")`
	Selects elements that contain a specific child element.
	Example: Selects all `<div>` elements that contain a `<span>` element. For example, only the first of the following elements would be selected:
	`<div><span>Span ➥Text</span></div>`
	`<div>No Span Text</div>`

`$(":empty")` `$("div:empty")`	Selects elements that have no content. Example: Selects all `<div>` elements that have no content.
`$(":parent")` `$("div:parent")`	Inverse of `:empty`, selects elements that have at least some content. Example: Selects all `<div>` elements that have at least some content.

Selecting Elements by Hierarchy Positioning

```
$("div span");
  //selects <span> elements with a <div> ancestor
$("div.menu > span");
  //selects <span> elements whose immediate parent
  //is a <div> with class="menu"
$("label + input.textItem");
  //selects <input> elements with class="textItem"
➥that
  //are immediately preceded by a <label>
$("#menu ~ div");
  //selects all <div> elements that are siblings of
  //the element with id="menu"
```

An important set of jQuery selectors is the hierarchy selectors. These selectors allow you to select HTML elements based on the DOM hierarchy. This allows you to write dynamic code that is more content aware by only selecting elements based on parents, children, or other elements around them in the DOM tree. Table 4.4 shows some examples of hierarchy selectors.

Table 4.4 **Examples of Using Hierarchy jQuery Selectors**

Syntax/Example	Description
`$(" ancestor element")` `$("div span")`	Selects elements of a type that have an ancestor of type ancestor and match element. Example: Selects all `<span>` elements that have an ancestor that is a `<div>`. The `<div>` element does not need to be the immediate ancestor. For example, the following `<span>` element would still be selected: `<div><p>Some <span>Span ➥Text</span></p></div>`
`$("parent > child")` `$("div.menu > span")`	Selects elements with a specific parent type. The `>` indicates an immediate child/parent relationship. Example: Selects all `<span>` elements that have an immediate parent element that is a `<div>` with `class="menu"`.
`$("prev + next")` `$("label + input. ➥textItem")`	Selects elements immediately followed by a specific type of element. The `+` indicates the `prev` item must be immediately followed by the `next` item. Example: Selects all `<label>` elements that are immediately followed by an `<input>` element that has `class="textItem"`.

Table 4.4 **Continued**

Syntax/Example	Description

`$("prev ~ siblings")`
`$("#menu ~ div")`
	Selects elements that are after the `prev`, have the same parent, and match the siblings selector. The ~ indicates siblings.
	Example: Selects all `<div>` elements that are siblings of the element that has `id="menu"` and come after the `"#menu"` item in the DOM tree. Note that `<div>` elements that come before will not be selected. For example, only the last two elements that follow will be selected:
	`<div>...</div>`
	`<ul id="menu> ... </ul>`
	`<span> ... <span>`
	`<div> ... </div>`
	`<div> ... </div>`

By the way

It is always best to be as specific as possible when designing your jQuery selectors. For example, if you want to select all the span elements with `class="menu"` and these elements are only under the `<div>` element with `id="menuDiv"`, then `$("div#menuDiv .menu")` would be much more efficient than just `$("menu")` because it would limit the search to the `<div>` element before checking from the menu `class` attribute.

Selecting Elements by Form Status

```
$("input:checked");
  //selects <input> elements that are checked
$("option:selected");
  //selects <option> elements that are selected
$("#myForm :focus");
  //selects the element in the #myForm <form>
  //that currently has the focus
$("input:disabled");
  //selects <input> elements that are currently
➥disabled
```

An extremely useful set of selectors when working with dynamic HTML forms is the form jQuery selectors. These selectors allow you to select elements in the form based on the state of the form element. Table 4.5 shows some examples of form selectors.

Table 4.5 **Examples of Using Attribute jQuery Selectors**

Syntax/Example	Description
`$(":checked")` `$("input:checked")`	Selects elements with `checked` attribute `true`. Example: Selects all `<input>` elements that are currently in a checked state.
`$(":selected")` `$("option:selected")`	Selects elements with the `selected` attribute true. Example: Selects all `<option>` elements that are currently `selected`.
`$(":focus")` `$(":focus")`	Selects elements that are in focus in the form. Example: Selects all HTML elements that are currently in focus.

Table 4.5 **Continued**

Syntax/Example	Description
`$(":enabled")` `$("input:enabled")`	Selects enabled elements. Example: Selects all the `<input>` elements that are in the `enabled` state.
`$("disabled")` `$("input:disabled")`	Selects disabled elements. Example: Selects all the `<input>` elements that are in the `disabled` state.

Selecting Elements Based on Visibility

```
$("div:visible");
  //selects <div> elements that are currently
➥visible
$("div:hidden");
  //selects <div> elements that are currently hidden
```

If you are using visibility to control the flow and interactions of your web page components then using the visibility jQuery selectors makes it simple to select the HTML elements that are hidden or visible. Table 4.6 shows some examples of visibility selectors.

Table 4.6 **Examples of Using Attribute jQuery Selectors**

Syntax/Example	Description
`$(":visible")`	Selects visible elements
`$("div:visible")`	Example: Selects all `<div>` elements currently have at least some height and width meaning the consume space in the browser. This will even include elements that are hidden by `visibility:hidden` or `opacity:0` because they still take up space.
`$(":hidden")`	Selects hidden elements.
`$("div:hidden")`	Example: Selects all `<div>` elements that currently have the CSS property of `visibility:hidden` or `opacity:0`.

Applying Filters in jQuery Selectors

```
$("tr:even"); //selects the even <tr> elements
$("li:odd"); //selects the odd <li> elements
$("div:first"); //selects the first <div> element
$("div:last"); //selects the last <div> element
$(":header"); //selects <h1>, <h2>, ... elements
$("div:eq(5)"); //selects the 6th <div> element
$("li:gt(1)"); //selects <div> elements after the
➥first two
$("li:lt(2)"); //selects the first two <div>
➥elements
$(":animated"); //selects elements currently
➥animating
```

Often you need to refine your jQuery selectors to a more specific subset. One way to accomplish that is to use filtered selectors. Filtered selectors append a filter on the end of the selector statement that limits the results that the selector returns. Table 4.7 shows some examples of adding filters to selectors.

Table 4.7 **Examples of Using Filtered jQuery Selectors**

Syntax/Example	Description
`$(":even")` `$("tr:even")`	Filters out all the odd indexed elements. Example: Selects all `<tr>` elements and then filters the results down to only the even numbered items.
`$("odd")` `$("li:odd")`	Filters out all the even indexed elements. Example: Selects all `<li>` elements and then filters the results down to only the odd numbered items.
`$(":first")` `$("div:first")`	Filters out everything but the first element. Example: Selects only the first `<div>` element encountered.
`$(":last")` `$("div:last")`	Filters out all but the last element. Example: Selects only the last `<div>` element encountered.
`$(":header")` `$(":header")`	Selects elements that are header types such as `<h1>`, `<h2>`, `<h3>`, and so on. Example: Selects all header elements.
`$("eq(index)")` `$("div:eq(5)")`	Selects the element at a specific zero-based index. Example: Selects the sixth `<div>` element encountered. The reason that the sixth element and not the fifth is selected is that the index is zero based, so 0 would be the first.

Syntax/Example	Description
`$("li:gt(index)")` `$("li:gt(1)")`	Filters the list to only include elements after a specific zero-based index. Example: Selects every `<li>` element after the second one encountered. Once again, this index is zero based.
`$(":lt(index)")` `$("li:lt(2)")`	Filters the list to only include elements before a specific zero-based index. Example: Selects only the first and second `<li>` elements encountered. Once again, this index is zero based.
`$(":animated")` `$(":animated")`	Selects elements that are currently being animated. Example: Selects all elements that are currently being animated.

Chaining jQuery Object Operations

```
$("div#content").children("p:first").css("font-
➥weight",
  "bold").children("span"). css("color","red");
```

One of the great things about jQuery objects is that you can chain multiple jQuery operations together into a single statement. Each consecutive statement operates on the results of the previous operation in the chain. This can simplify your selectors as well as reduce the amount of class and ID definitions required in your CSS.

To help illustrate this, consider the following statements. The code first finds the `<div>` element with

id="content" and then finds the first <p> element inside and changes the font-weight to bold. Then it finds the elements inside the <p> and sets the color to red:

```
var $contentDiv = $("div#content");
var $firstP = $contentDiv.children("p:first");
$firstP.css("font-weight","bold");
var $spans = $firstP.children("span");
$spans.css("color","red");
```

The previous code took five lines to accomplish all its tasks. The following statement of chained jQuery operations does the same things but with only a single statement:

```
$("div#content").children("p:first").css("font-
➥weight","bold").children("span").
css("color","red");
```

Because each of the operations returns a jQuery object, you can chain as many jQuery operations together as you would like. Even though the .css() operation is designed to alter the DOM objects and not find them, it still returns the same jQuery object so you can perform other operations on the results.

Navigating jQuery Objects to Select Elements

Another important set of methods attached to a jQuery object is the DOM traversing methods. DOM traversal allows you to select elements based on their relationship to other elements. A couple of examples of DOM traversal are accessing all <p> elements that are

children of elements and finding a <label> element that is a sibling of an <input> element.

jQuery object provides an incredible set of methods that allow you to traverse the DOM in almost innumerable ways by allowing you to use the current selection of DOM elements in the jQuery object to select other sets of DOM elements in the tree. The following phrases provide some examples of traversing the DOM elements in jQuery objects to get to other sets of objects.

Getting the Children of Elements

```
$("div").children("p");
  //selects the <p> elements that are direct
➥children
  //of <div> elements
```

The .children([selector]) method returns a jQuery object representing the children of the elements represented by the current object. You can specify an optional selector that limits the results to only include children that match the selector.

Getting the Closest Elements

```
$("p.menu").closest("div");
  //selects the closest <div> ancestor for <p>
  //elements that have class="menu"
```

The .closest(selector,[context] or object or element) method returns a jQuery object representing the first element that matches the argument that is passed in. The argument can be a selector, a selector with context of where to begin searching, a jQuery object, or a DOM object. The search begins at the current set of elements and then searches ancestors.

Getting the Elements Contained

```
$("div").contents();
  //selects all the immediate child elements in
➥<div> elements
$("select").contents();
  //selects all the <option> elements in <select>
➥elements
```

The `.contents()` returns a jQuery object representing the immediate children of the current set of elements. This is especially useful when getting all the `<li>` elements in a list or the `<option>` elements in a `<select>` block.

Finding Descendant Elements

```
$("table").find("span")
  //selects all <span> elements contained
  //somewhere in <table> elements
$("#myForm").find("input")
  //selects all <input> elements contained
  //somewhere in element #myForm
```

The `.find(filter)` method returns a jQuery object representing descendants of the current set that match the `filter` supplied. The `filter` can be a selector, a jQuery object to match elements against, or an element tag name.

Example: Selects all `<span>` elements contained somewhere in `<table>` elements.

Getting Siblings That Come After Selected Objects

```
$("#title").next("p");
  //finds the element with id="title" and selects
  //the very next <p> element that is a sibling
$("p:first").nextAll();
  //selects the first <p> element that it finds and
  //then selects all the <p> siblings to that
➥element
$("p:first").nextUntil("ul");
  //selects the first <p> element that it finds and
  //then selects all the siblings until it finds a
➥<ul> element
```

The `.next([selector])` method returns a jQuery object representing the next sibling of each element in the current set. If the optional `selector` is provided, the next sibling is added only if it matches the `selector`.

The `.nextAll([selector])` method returns a jQuery object representing all the following sibling objects of each element in the current set. Also accepts an optional `selector`.

The `.nextUntil([selector] or [element] [,filter])` method returns a jQuery object representing all the sibling objects after each element in the current set, until an object matching the element or selector argument is encountered. The first argument can be a DOM `object` or a `selector`. The second optional argument is a `filter` selector to limit the items returned in the results.

Getting the Positioning Parents

```
$("#notify").offsetParent();
  //selects the element with id="notify" and then
➥selects
  //the ancestor that is used to position that
➥element
```

The `.offsetParent()` method returns a jQuery object representing the closest ancestor element that has a CSS `position` attribute of `relative`, `absolute`, or `fixed`. This allows you to get the element used for positioning, which becomes critical when you need to get the size of the position container.

Finding the Parent or Ancestors of Selected Items

```
$("div#menu").parent();
  //selects the <div> element with id="menu" and
  // then finds its immediate parent
$("#data").parents("div");
  //selects the element with id="data" and then
  //returns a set with all <div> ancestors
$("#data").parentsUntil("#menu");
  //selects the parents of #data until
  //it finds the one with id="#menu"
```

The `.parent([selector])` method returns a jQuery object representing the immediate parent objects of each of the elements represented in the current set. An optional selector argument allows you to limit the results to those parents matching the selector.

The `.parents([selector])` method returns a jQuery object representing the ancestors of each of the elements represented in the current set. An optional `selector` argument allows you to limit the results to those parents matching the `selector`.

The `.parentsUntil([selector] or [element] [, filter])` method returns a jQuery object representing the ancestors of each of the elements represented in the current set, until an object matching the element or selector argument is encountered. The first argument can be an `element` tag name or a `selector`. The second optional argument is a `filter` selector to limit the items returned in the results.

Getting the Previous Siblings

```
$("p#footer").prev("p")
  //Finds the <p> element with id="footer" and
  //selects the previous <p> element that is a
➥sibling
$("div#footer").prevAll("div");
  //selects the <div> element with id="footer" and
➥then
  //selects all the <div> siblings prior to that
➥element
$("div#footer").prevUntil("div#header");
  //finds the <div> element with id="footer" and
➥then
  //selects all prior siblings until #header is
➥found
```

The `.prev([selector])` method returns a jQuery object representing the previous sibling of each element in the current set. If the optional `selector` is provided, the previous sibling is added only if it matches the `selector`.

The `.prevAll([selector])` method returns a jQuery object representing all the previous sibling objects of each element in the current set. It also accepts an optional `selector`.

The `.prevUntil([selector]` or `[element] [,filter])` method returns a jQuery object representing all the sibling objects that come before each element in the

current set, until an object matching the element or selector argument is encountered. The first argument can be a DOM `object` or a `selector`. The second optional argument is a `filter` selector to limit the items returned in the results.

Getting Siblings

```
$(".menu").siblings("span")
  //selects all <span> elements that are
  //siblings to elements with class="menu"
```

The `.siblings([selector])` method returns a jQuery object representing all the sibling objects for each element in the current set. An optional `selector` argument allows you to limit the results to those siblings matching the `selector`.

5

Manipulating the jQuery Object Set

An important aspect of jQuery objects is the ability to access and manipulate the set of DOM elements that are represented by the object. This is different than manipulating the DOM elements themselves.

jQuery provides several methods that allow you to modify which DOM elements are represented by the jQuery object. There are also methods to iterate through each element to build different elements or apply other operations.

The following phrases are designed to show you how to work with the jQuery object set in various ways.

Getting DOM Objects from a jQuery Object Set

```
$("div").get();
  //returns an array of DOM object for all <div>
➥elements
$("div").get(0);
  //returns a DOM object for the first <div>
$("div").get(-1);
  //returns a DOM object for the last <div>
```

The `.get([index])` method returns the DOM elements represented in the jQuery object set. If no `index` is specified, an array of the DOM objects is returned. If an `index` is specified, the DOM element at that zero-based offset in the array is returned.

If the `index` is a negative number, the item at the reverse offset from the end of the array is returned. For example, –1 is the last item and –2 is the second to the last item.

Converting DOM Objects into jQuery Objects

```
var containerObj = document.getElementById("myDiv");
$(containerObj); //Equivalent to $("#myDiv");
var objs =
➥document.getElementsByClassName("myClass");
$(objs); //Equivalent to $(".myClass ");
var objs = document.getElementsByTagName("div");
$(objs); //Equivalent to $("div");
```

The jQuery constructor `$()` accepts DOM objects and arrays as an argument. If you pass in a DOM object, `$()` converts the DOM object or array into a jQuery object that you can then manipulate in the normal jQuery way.

Iterating Through the jQuery Object Set Using .each()

```
$("p").each(function (idx){
    $(this).html("This is paragraph " + idx);
  });
```

The .each(function) method is one of the most important jQuery object methods because it allows you to traverse all elements in the jQuery set and perform actions on each of them individually. This is different from just applying the same action to all items in the query.

The .each() method allows you to specify a function to run for each element in the jQuery object set. The function is passed an index number as the first argument. Inside the function, the this variable points to the current DOM element.

To illustrate using .each(), check out the following snippet of code. It iterates through all paragraph elements and sets the content of the empty <p> elements to a string that includes the index number of the element, as shown in Figure 5.1:

```
01 <html>
02 <head>
03 <title>Python Phrasebook</title><meta
➥charset="utf-8" />
04 <script type="text/javascript" src="../js/jquery-
➥2.0.3.min.js"></script>
05 <script>
06   $(document).ready(function (){
07     $("p").each(function (idx){
08         $(this).html("This is paragraph " + idx);
09       });
```

```
10   });
11 </script>
12 </head>
13   <body>
14     <p></p>
15     <p></p>
16     <p></p>
17     <p></p>
18   </body>
19 </html>
```

ch0501.html

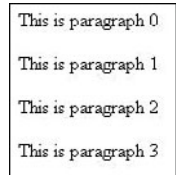


Figure 5.1 Results of running the ch0501.html code.

Notice that `idx` is passed in as an index number, 0 for the first `<p>` element, 1 for the second, and so on. Also note that I converted `this` to a jQuery object using `$(this)` so that I could call the `.html()` method.

By the way

When using functions with jQuery methods that iterate through the DOM elements, you can use the `this` keyword to access the current DOM object that is being iterated on. `this` is a DOM object and not a jQuery object, so if you need to use the jQuery form of the DOM object, use `$(this)`. Keep in mind, though, that it takes work to build the jQuery form of the DOM object, so only create the jQuery form if you want the functionality that is provided.

Using .map()

```
var liValues = $("li").map(function (idx){
  return $(this).html();
}).get();
```

The .map(function) method also iterates through each element in the jQuery object set. Although similar to .each(), there is one big difference: .each() returns the same jQuery object, but .map() returns a new jQuery object with the values returned by each iteration.

To illustrate using .map(), check out the following snippet of code. This code iterates through all <li> elements and returns an array containing the HTML content values inside the elements shown in Figure 5.2:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    $(document).ready(function (){
      var liValues = $("li").map(function (idx){
        return $(this).html();
      }).get();
    });
  </script>
</head>
  <body>
    <ul>
      <li>Aragorn</li>
      <li>Legolas</li>
      <li>Gimli</li>
```

```
20      </ul>
21    </body>
22 </html>
```

ch0502.html

```
[
 "Aragorn",
  "Legolas",
  "Gimli"
]
```

Ending value of `liValues`

Figure 5.2 Results of running the ch0502.html code.

Notice that during each iteration, the function returns the HTML content in the `<li>` element. This value is added to the mapped object set that the `.map()` method returns. The `.get()` call at the end gets the JavaScript array object represented in the mapped set.

Assigning Data Values to Objects

```
$("li:eq(0)").data("race","Men");
$("li:eq(1)").data("race","Elves");
$("li:eq(2)").data({race:"Dwarves"});
$("li").each(function(){
  $(this).append(" of the race of ");
  $(this).append($(this).data("race"));
});
```

jQuery provides `the .data(key [,value])` method to store pieces of data as `key=value` pairs inside the DOM object. This allows you to attach additional pieces of data to the object that you can access later.

There is an alternate form of `.data(object)` that accepts JavaScript objects containing the key value pairs. This becomes useful when you are already working with objects.

To access a data item once it has been stored, simply call `.data(key)`, which returns the object `value`. You can attach data to any kind of HTML object. For instance, you can attach additional data to select `<option>` elements that provide more information than the traditional values support.

The following code shows an example of storing data in objects. Line numbers 9–11 store data in `<li>` elements. Then each function in lines 12–15 iterates through the `<li>` elements and appends a string with the stored value onto the content. Figure 5.3 shows the resulting HTML page:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    $(document).ready(function (){
      $("li:eq(0)").data("race","Men");
      $("li:eq(1)").data("race","Elves");
      $("li:eq(2)").data("race","Dwarves");
      $("li").each(function(){
        $(this).append(" of the race of ");
        $(this).append($(this).data("race"));
      });
    });
  </script>
</head>
  <body>
    <ul>
      <li>Aragorn</li>
      <li>Legolas</li>
      <li>Gimli</li>
    </ul>
  </body>
</html>
```

ch0503.html

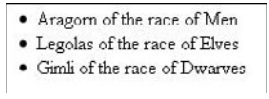


Figure 5.3 Results of running the ch0503.html code.

Adding DOM Elements to the jQuery Object Set

```
var pars = $("p");
  //contains all <p> elements
var pAndDiv = $("div").add(pars);
  //contains all <div> and <p> elements
var pAndDivAndSpan = $("span").add("p").add("div");
  //contains all <span> and <div> and <p> elements
```

A great feature of jQuery is how easy it is to add additional elements to the jQuery object set. The `.add(selector)` method adds elements that match the `selector` to the set represented by the jQuery object.

The `.add(object)` method accepts another jQuery `object` and adds the elements in that object to the current object's set.

Removing Objects from the jQuery Object Set

```
var items = $("span");
items.css({color:"red"});
  //colors text in all <span> elements red
items.remove(".menu"):
items.css({"font_weight":"bold"});
  //set all <span> elements except .menu to bold
```

Another great feature of jQuery is how easy it is to remove items from and add additional elements to the jQuery object set. The `.remove([selector])` method will remove elements that match the `selector` from the set represented by the jQuery object. Calling `.remove()` with no selector will remove all elements from the set.

Filtering the jQuery Object Results

jQuery objects provide a good set of methods that allow you to alter the DOM objects represented in the query. Reducing the results is helpful when you are trying to pinpoint a specific set of elements within a jQuery selector result. The following phrases provide examples of filtering the results of a jQuery selector to find the set of objects.

Filtering Items Based on Index

```
$("div#content").eq(1);
  //selects the second element found by the selector
```

The `.eq(index)` filter will return the element at a specific index in the jQuery object's set. The index is zero based, so `.eq(1)` actually retrieves the second element found by the selector.

Filtering Selector Results Using a Filter Function

```
$("option").filter(
 function (index) {
  return (this.value>5); });
  //selects only the <option> elements with value=5
$("div").filter(".myClass");
  //selects only the <div> elements with
➥class="myClass"
```

The `.filter(filter)` method reduces the set to only those items matching the specified filter. The filter can be a selector, a specific DOM element, a jQuery object, or a function that tests each element and returns `true` if it should be included.

Did you know?

The jQuery selectors that are the same as the CSS selectors are able to use the native DOM method `querySelectorAll()`, which has some advanced optimizations on DOM objects. Other jQuery selectors cannot take advantage of that optimization, so it is better to use a CSS-based selector first and then add the filter as a chained selector. For example, rather than using `$("div:animated")`, you should use `$("div").filter(":animated")`.

Getting the First or Last Item

```
$("p").first(); //selects the first <p> elmemnt in
➥the set
$("p").last(); //selects the last <p> elmemnt in the
➥set
```

The `.first()` method selects the first element in the jQuery object's set. Many of the jQuery methods that return values end up using the first element in the set. The `.last()` method selects the last element in the jQuery object's set.

Getting Items That Contain Other Items

```
$("div").has("p");
  //selects <div> element that has <p> descendents
$("div").has("#menu");
  //selects <div> element that contains
  //the element that has id="menu"
```

The `.has(selector or element)` method reduces the set to those elements that have descendent elements that match the selector or contain the specified DOM element.

Filter Items Out Using Not Logic

```
$("div").not(".menu");
  //selects <div> element; do not have class .menu
$("option").not(function(){
  return (this.value==0); });
});
  //selects <option> elements that do not have
➥value=0
```

The `.not(filter)` method reduces the set to match the `filter`. The `filter` can be a selector, one or more specific DOM elements, a jQuery object, or a function that tests each element and returns `true` if it should be included.

Slicing Up Selector Results

```
$("tr").slice(2,5);
  //selects the <tr> elements between 2 and 4
➥inclusive
$("p").slice(1,-1);
  //selects all <p> elments except the first and
➥last
```

The `.slice(start, [end])` method removes elements before index `start` and after index `end` and returns only the items in between. This allows you to break large selections into smaller chunks. The indexes are zero based. A negative `end` indicates the offset from the end of the list, where –1 is the last element and –2 is the second to the last element.

6

Capturing and Using Browser and User Events

One of the major goals of jQuery and JavaScript is to allow developers to create incredibly sophisticated and richly active web pages. At the heart of interactive web pages are events. An *event* refers to anything happening in the browser environment, from a page loading, to a mouse movement or click, to keyboard input, to resizing of the window.

Understanding events, the objects that represent them, and how to apply them in your web pages will enable to you to create some spectacular user interaction. This chapter covers the concepts required to understand events and how to utilize them in building rich, interactive web pages.

The phrases in this chapter are designed to show you how the JavaScript event engine works and how to apply JavaScript and jQuery code to implement event handlers. You will be able to apply the concepts in this chapter with concepts from other chapters to implement professional and rich user interactions.

Understanding Events

The browser event concept is pretty simple. An event is anything that alters the state of the browser or web page. These include loading of the page, mouse movements and clicks, keyboard input, as well as form submissions and other things occurring in the page. Each event has an event object that describes the event that took place and what object it took place on.

This section provides a few tables to help you get a quick overview of what event objects look like, what some of the more common events triggered in JavaScript are, and what triggers them.

Reviewing Event Types

An event type is a keyword that JavaScript and jQuery use to identify the physical event that took place. These keywords are used to attach handlers to specify types of events. Also, as you will see later on, the keywords are used in the names of methods so that you can attach event handlers or trigger events manually.

JavaScript provides several event types that correspond to the different events that occur during dynamic page interaction. jQuery event handling supports all the JavaScript and adds a few event types of its own. Table 6.1 lists the different event types that JavaScript and jQuery support.

Table 6.1 **JavaScript and jQuery Event Types**

Event Type	Description
`abort`	Triggered when an image load is stopped before completing.
`blur`	Triggered when a form element loses focus.
`change`	Triggered when the content, selection, or check state of a form element changes. Applies to `<input>`, `<select>`, and `<textarea>` elements.
`click`	Triggered when the user clicks on an HTML element.
`dblclick`	Triggered when the user double-clicks on an HTML element.
`error`	Triggered when an image does not load correctly.
`focus`	Triggered when a form element gets the focus.
`focusin`	Triggered when a form element or any element inside of it gets the focus. Different from `focus` in that it supports bubbling up to parents. This is a jQuery-only event.
`focusout`	Triggered when a form element or any element inside of it loses the focus. Different from `blur` in that it supports bubbling up to parents. This is a jQuery-only event.
`keydown`	Triggered when a user is pressing a key.
`keypress`	Triggered when a user presses a key.
`keyup`	Triggered when a user releases a key.
`load`	Triggered when a document, frameset, or DOM element is loaded.

Table 6.1 **Continued**

Event Type	Description
`mousedown`	Triggered when a user presses a mouse button. The target element is the one the mouse is over when the button is pressed.
`mousemove`	Triggered when the mouse cursor is moving over an element.
`mouseover`	Triggered when the mouse cursor moves onto an element or any of its descendants.
`mouseout`	Triggered when the mouse cursor moves out of an element or any of its descendants.
`mouseup`	Triggered when a user releases a mouse button. The target element is the one that the mouse is over when the button is released.
`mouseenter`	Triggered when the mouse cursor enters an element. Different from `mouseover` in that it is triggered only by the element and not its descendants. This is a jQuery-only event.
`mouseleave`	Triggered when the mouse cursor leaves an element. Different from `mouseout` in that it is triggered only by the element and not its descendants. This is a jQuery-only event.
`reset`	Triggered when a form is reset.
`resize`	Triggered when the document view is resized by resizing the browser window or frame.

Event Type	Description
`scroll`	Triggered when a document view is scrolled.
`select`	Triggered when a user selects text in an `<input>` or `<textarea>`.
`submit`	Triggered when a form is submitted.
`unload`	Triggered when a page is unloaded from the `<body>` or `<frameset>` element.

Adding Event Handlers

An *event handler* is a function that is called when the event is triggered. The event object is passed to the event handler so you will have access in the event handler to information about the event.

The following phrases show ways to add event handlers in HTML, JavaScript, and jQuery.

Adding a Page Load Event Handler in JavaScript

```
<script>
  function onloadHandler(){
    (initialization code here...)
  }
</script>
...
<body onload="onloadHandler()>
```

To add initialization code that runs when the pages are loaded in JavaScript, create a function in JavaScript that performs the initialization. For example, the following JavaScript code shows a simple skeleton initialization function:

```
function onloadHandler(){
  (initialization code here...)
}
```

To cause the `onloadHandler()` to trigger when the page is fully loaded, add the event hander function to the `onload` attribute of the `<body>` element in the HTML. For example:

```
<body onload="onloadHandler()>
```

Adding Initialization Code in jQuery

```
$(document).ready(function(){
  (initialization code here...)
}
or
$(document).load(function(){
  (initialization code here...)
}
```

In jQuery, you can trigger and execute initialization code at two different times: when the DOM is ready, and when the document and its resources have fully loaded. The option to use depends on what needs to happen in your initialization code.

Using the `.ready()` jQuery method triggers the initialization code to run when the DOM is fully ready. All the DOM objects will be created, and the page will be displayed to users. Note that not all page resources, such as images, may have fully downloaded at this point. This is the option that I use most frequently because it allows me to add interactions and functionality as soon as possible. The following shows an example of using `.ready()` to attach a simple initialization function:

```
$(document).ready(function(){
  (initialization code here...)
}
```

Using the `.load()` jQuery method triggers the initialization code to run after all page resources have loaded and are rendered to the user. On occasion, I use this option if I need resource information, such as image dimensions, when I am applying the initialization code. The following code shows an example of using `load()` to attach a simple initialization function:

```
$(document).load(function(){
  (initialization code here...)
}
```

Watch out!

The `.ready()` and `.load()` methods are not compatible with using the `onload="..."` attribute in the `<body>` tag. If you are using jQuery, use `.ready()` or `.load()`; if not, use the `onload` attribute.

Assigning an Event Handler in HTML

```
function clickHandler(e){
  $("div").html("clicked at X postion: " +
➥e.screenX);
}
. . .
<div onclick="clickHandler(event)">Click Here</div>
```

The most basic method of adding an event handler to a DOM element is directly in the HTML code. The advantage of this method is that it is simple and easy to

see what event handler is assigned to a particular DOM object.

Event handlers are added to DOM objects in HTML by setting the value of the handler attribute in the `tag` statement. For each event that the element supports, there is an attribute that starts with `on` followed by the name of the event. For example, the `click` event attribute is `onclick`, and the `load` event attribute is `onload`. To view a list of the event types, see Table 6.1.

The following example shows just how easy it is to add a `click` event handler to a DOM element in the HTML code:

```
<div onclick="clickHandler()">Click Here</div>
```

The browser will call the following function when you click the `<div>` element:

```
function clickHandler(){
  $("div").html("clicked");
}
```

You can also include the DOM event object as a parameter to the event handler using the `event` keyword. This allows you to access the event information in the event handler, for example:

```
<div onclick="clickHandler(event)">Click Here</div>
```

The browser will call the following function when the `<div>` element is clicked and change the text to display the `x` coordinate of the mouse cursor by reading `e.screenX`:

```
function clickHandler(e){
  $("div").html("clicked at X postion: " +
➥e.screenX);
}
```

Adding Event Handlers in JavaScript

```
function clickHandler(e, data){
  (Event Handler Code)
}
...
document.getElementById("div1").addEventListener
➥('click',
  function(e){
    eventHandler(e,"data");
  },false);
```

You can also dynamically add and remove event handlers to DOM objects inside JavaScript code. To add an event handler in JavaScript, simply call `addEventListener()` on the DOM object.

The `addEventListener()` function takes three parameters. The first is the event `type` (event types are defined in Table 6.1), a function to call and a boolean that specifies `true` if the handler should be called during the capturing phase and the bubbling phase or `false` if the handler should be called only during the bubbling phase.

To pass custom data into the handler call, wrap the actual function handler inside of a simple wrapper function that passes the arguments to the event handler.

For example, the following code calls the click handler with additional information, including the ID of a different object that is updated:

```
function clickHandler(e,objId,num,msg){
  var obj = document.getElementById(objId);
  obj.innerHTML = "DIV " + num + " says " + msg +"
➥at X postion: " + e.screenX;
}
...
document.getElementById("div1").addEventListener
➥('click',
  function(e){
    eventHandler(e, "heading", 1, "yes");
  },false);
```

By the way

You can also set the access event handler on the DOM object using the handler attribute. The handler attribute will be "on" plus the event type. For example, for the `click` event, the attribute is `obj.onclick`. You can call the event handler using `obj.onclick()` or assign it directly using `obj.onclick= function handler(){ ...};`.

Removing Event Handlers in JavaScript

```
var obj = document.getElementById("div1");
obj.addEventListener('click', clickHandler);
. . .
obj.removeEventListener('click', clickHandler);
```

You can also remove event handlers from the event using the `removeEventListener()` function on the DOM object.You need to specify the event type and the name of the event handler to remove. For example, the following code adds an event handler and then removes it:

```
function clickHandler(e){
   . . .
}
var obj = document.getElementById("div1");
obj.addEventListener('click', clickHandler);
```

Adding Event Handlers in jQuery

```
$("div").on("click", "span", {name:"Brad",
➥number:7},
  function(e){
    (event handler code here)
}):
```

Events are added in jQuery using the `.on(events,[,selector][,data]handler(eventObject)` method. This method accepts one or more `event` types such as those listed in Table 6.1 as the first argument.

The optional `selector` value defines the descendant elements that trigger the event. Any descendant elements that match the selector also trigger the event.

The `data` object is anything that you want to pass to the event handler in the `event.data` attribute. Typically, this is either a single value or a JavaScript object. However, anything is accepted.

The `handler` function is the function to be executed when the event is triggered. The `eventObject` is passed to the handler function and provides you with the details about the event.

The great thing about using jQuery to add event handlers is that you get to use the jQuery selector functionality. For example, to add the `click` handler to the first paragraph in the document, you would use the following:

```
$("p:first").on("click", function(e){ (event handler
➥code here) }):
```

Or to add a `keyup` event handler to all `<input>` elements with the `.texting` class set, you would use this:

```
$("input.texting").on("click", function(e){ (event
➥handler code here) }):
```

Another example that adds a `click` event handler to all `<div>` elements that is also triggered on `<span>` descendants and passes an object follows:

```
$("div").on("click", "span", {name:"Brad",
➥number:7},
  function(e){
    (event handler code here)
}):
```

Removing Event Handlers in jQuery

```
$("div").off("keypress"):
or
$("div").off("click", "span",
  function(e){
    (event handler code here)
}):
```

Removing an event handler in jQuery is extremely easy using the `.off(events[,selector ][,handler (eventObject)])` method. This method has similar arguments to the `.on()` event handler. The arguments match so closely so that you can add multiple event handlers to elements. Using the arguments helps jQuery know which event matches the one you want to turn off.

Controlling Events

An important aspect of handling JavaScript is the ability to control the event behavior. The following list describes the event process that happens when a user interacts with the web page or browser window.

1. **Physical event happens**—A physical event occurs. For example, a user clicks or moves the mouse or presses a key.
2. **Events are triggered in the browser**—The user interaction results in events being triggered by the web browser. Often, multiple events are triggered at the same time. For example, when a user presses a key on the keyboard, three events are triggered: `keypressed`, `keydown`, and `keyup`.
3. **Browser creates an object for the event**—The web browser creates a separate object for each event that is triggered. The objects contain information about the event that handlers can use.
4. **User event handlers are called**—User-defined event handlers are called. You can create handlers in JavaScript that interact with the event objects or page elements to provide interactivity with HTML elements. The event handlers can be acting in three phases. The following describes the three phases shown in Figure 6.1:
    - **Capturing**—The capturing phase occurs on the way down to the target HTML element from the document directly through each of the parent elements. By default, behavior for event handlers for the capturing phase is disabled.

- **Target**—The target phase occurs when the event is in the HTML element where it was initially triggered.
- **Bubbling**—The bubbling phase occurs on the way up through each of the parents of the target HTML element all the way back to the document. By default, the bubbling phase is enabled for events.

5. **Browser handlers are called**—In addition to user event handlers, the browser has default handlers that do different things based on the event that was triggered. For example, when the user clicks on a link, the browser has an event handler that is called and navigates to the `href` location specified in the link.

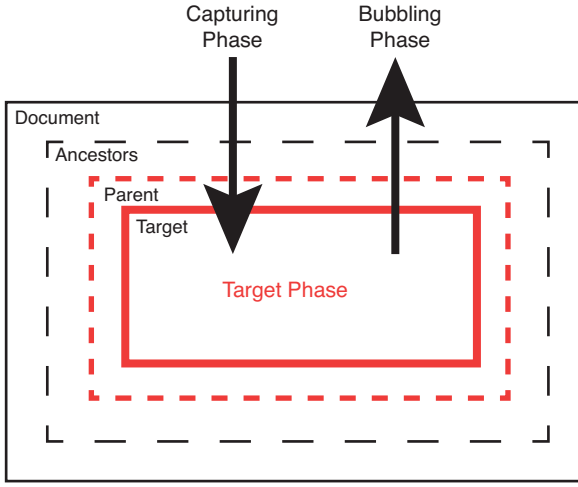


Figure 6.1 Events are handled first in the capturing phase from the document down to the target, then in the target phase, and finally in the bubbling phase from the target up through the document.

The following phrases describe ways to interact with and control the event process.

Stopping Event Bubbling Up to Other Elements

```
function myEventHandler(e){
  ...handler code...
  e.stopPropogation();
  return false;
}
```

Often, you don't want an event to bubble up to parent objects because you have already handled it in a child object. JavaScript provides a simple way to stop bubbling from occurring. Calling the `event.stopPropagation()` method on the event object passed to the handler stops the event from bubbling up to parent elements. The event still triggers the default handlers in the browser that provide the web page interaction, such as selecting a check box or activating a link.

Stopping Default Behavior

```
function myEventHandler(e){
  ...handler code...
  e.preventDefault();
  return false;
}
```

Occasionally, you might want to prohibit the browser from applying the default handler interaction for the event. For example, you might not want the browser to activate a link, or you might not want to allow a form to be reset. You can stop the default browser action by calling the `event.preventDefault()` method on the event object passed to the handler.

Triggering Events Manually in jQuery

```
$("span").mouseenter();
$("#myButton").click();
$("input.bigText").trigger({'type':'keypress',
➥'charCode':13});
```

jQuery objects have methods such as `click()` and `dblclick()` that correspond to many of the event types that you can call directly. Form elements add additional methods such as `blur()`, `focus()`, `keypress()`, `keydown()`, and `keyup()` that can be called to trigger specific events. For example, the following statement triggers the `click` event for all `<span>` elements:

```
$("span").click();
```

jQuery also provides a way to trigger events while specifying the values of the event object using the `trigger()` method. There are two different syntaxes for the `trigger()` method, as listed here:

```
trigger(eventType [, extraParameters])
trigger( eventObject)
```

The following is an example of using the first method to trigger the `click` event for all elements with `class="checkbox"`:

```
$(".checkbox").trigger("click");
```

Next is an example of using the second method on all input items with `class="bigText"`. This method actually passes in a simple event object for the `keypress` event that sets the `charCode` attribute of the event object to 13 or the Return key.

```
$("input.bigText").trigger({'type':'keypress',
➥'charCode':13});
```

Using Event Objects

The browser creates event objects when it detects that an event has occurred. If jQuery defined the event handler, the event object is converted to a jQuery event object that has a few more attributes and methods. Therefore, you need to be aware of which event object type you are working with.

Event objects provide additional information about the event, such as what type the event was (such as a `click` or a `keypress`), which key(s) were pressed, what position the mouse was in, what HTML element the event occurred on, and so on. Table 6.2 describes the most commonly used event attributes that you will be working with.

Table 6.2 **JavaScript and jQuery Event Object Attributes**

Property	Description
`altKey`	`true` if the Alt key was pressed during the event; otherwise, `false`.
`button`	Returns the number of the mouse button that was pressed: 0 for left, 1 for middle, and 2 for right.
`cancelable`	`true` if the default action for the event can be stopped; otherwise, `false`.
`charCode`	Ordinal value of the character that was pressed if it is a keyboard event.

Table 6.2 **Continued**

Property	Description
`clientX`	Horizontal coordinate of the mouse pointer relative to the current window.
`clientY`	Vertical coordinate of the mouse pointer relative to the current window.
`ctrlKey`	`true` if the Ctrl key was pressed during the event; otherwise, `false`.
`currentTarget`	The DOM object for the HTML element that the event handler currently being executed is attached to.
`data`	User data defined when the event object is created and is attached to the event object passed to the event handler when the event is triggered.
`delegateTarget`	The DOM object of the HTML element that was used in the jQuery `.delegate()` or `.on()` method to attach the event handler. (`.on()` and `.delegate()` are discussed later in this chapter.) Available only on jQuery event objects.
`eventPhase`	The current phrase that the event handler is operating in, where 1 is at the target, 2 is bubbling, and 3 is capturing.
`metaKey`	`true` if the "meta" key was pressed during the event; otherwise, `false`.

Property	Description
`relatedTarget`	Identifies a secondary target relative to the UI event. For example, when using the `mouseover` and `mouseexit` events, this indicates the target being exited or entered.
`results`	Last value returned by an event handler that was triggered by this event. Available only on jQuery event objects.
`screenX`	Horizontal coordinate based on the actual display coordinate system.
`screenY`	Vertical coordinate based on the actual display coordinate system.
`shiftKey`	`true` if the Shift key was pressed during the event; otherwise, `false`.
`target`	The DOM object for the HTML element where the event originated.
`type`	Text describing the event, such as `click` or `keydown`.
`timeStamp`	Time in ms since January 1, 1970, and when the event was triggered.
`which`	Actual numerical value.

The following phrases provide examples of utilizing the data in the event object.

Getting the Event Target Objects

```
function myHandler(e){
  //get the value of the object that triggered the
➥event
  var originalValue = $(e.target).val();
  //get the value of the current object
  var currentValue = $(e.currentTarget).val();
  //get the inner HTML of the current object
  var currentHTML = $(this).html();
}
```

The `event` object provides a link to the DOM object that originally triggered the event as the event `.target` attribute. The `event` object also provides a link to the DOM object that triggered the current event handler as the `event.currentTarget` attribute. Having access to these objects is useful because it allows you to obtain additional information, such as `checked` state or `value`.

The object that triggered the event handler is also available using the `this` keyword. For example, you can get the inner HTML value of the element using the following:

```
var currentHTML = $(this).html();
```

Getting the Mouse Coordinates

```
function myHandler(e){
  $("#p1").html("From left side of screen: " +
➥e.screenX);
  $("#p2").html("From right side of screen: " +
➥e.screenX);
  $("#p3").html("From left side of browser: " +
➥e.clientX);
  $("#p4").html("From right side of browser: " +
➥e.clientY);
}
```

For many graphical web apps, it is critical to not only know what target mouse events originated in, but the exact location on the screen or in the web browser display. The JavaScript mouse event object includes the `event.screenX` and `event.screenY` attributes that allow you to get the offset from the top and left of the screen in pixels.

Additionally, the JavaScript mouse event includes the `event.clientY` and `event.clientY` attributes that allow you to get the offset from the top and left of the browser window in pixels. These are often the more important coordinates because they help establish the mouse position in relation to the web page.

Handling Mouse Events

The most common events that occur in web browsers are mouse events. Each movement of the mouse is an event. Events also include a mouse entering or leaving an HTML element and mouse clicks and scrolls. The phrases in this section give you some examples of utilizing the mouse events in your JavaScript and jQuery code.

Adding Mouse-Click-Handling Code

```
$("p").on("click", function (e){
  $(this).html("You clicked here.");
});
$("span").on("dblclick", function (e){
  $(this).html("You double-clicked here.");
});
```

To register a `click` handler in jQuery, you should use the `.on("click")` or `.on("dblclick")` method. Inside the handler function, you can perform whatever event

functionality your application requires. It is easy to apply the `click` handler to multiple objects using jQuery selectors. For example, the following applies a click handler to all `<div>` elements:

```
$("div").on("click", function (e){
  $(this).html("You clicked here.");
});
```

Handling the Mouse Entering or Leaving Elements

```
$("p").on("mouseover", function (e){
  $(this).html("Hello mouse.");
});
$("p").on("mouseout", function (e){
  $(this).html("Wait, don't go.");
});
```

The browser provides an event each time the mouse enters or leaves an element. The standard JavaScript events are `mouseover` and `mouseout`, so you can use `.on("mouseover")` to add an event handler for the mouse entering an element and `.on("mouseout")` to add an event handler for when the mouse leaves.

Applying a Right-Click

```
$("#menu").hide();
$("#main").on("mousedown", function(e){
  if(e.button == 2){ $("#menu").show(); } });
$("#main").on("mouseup", function(e){
  if(e.button == 2){ $("#menu").hide(); } });
$('#main').on("contextmenu", function(e){
  return false; });
```

Applying a right-click to an element requires some additional actions because the default browser action is to bring up a right-click menu for the browser page.

The first thing you need to do in your event handler is to determine which mouse button was clicked by looking at the `event.button` attribute, which contains 0 for left, 1 for center, and 2 for right.

You also need to suppress the default browser `contextmenu` behavior, as shown in lines 14 and 15, in most instances to add your own right-click behavior. The following code example demonstrates how to implement the framework for your own custom popup menu that is displayed only when you right-click, as shown in Figure 6.2:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function(){
09       $("#menu").hide();
10       $("#main").on("mousedown", function(e){
11         if(e.button == 2){ $("#menu").show(); }
➥});
12       $("#main").on("mouseup", function(e){
13         if(e.button == 2){ $("#menu").hide(); }
➥});
14       $('#main').on("contextmenu", function(e){
15         return false; })
16     });
17   </script>
18 </head>
```

```
19 <body>
20   <div id="main"> Right-Click Me
21     <div id="menu">
22       <p>Item 1</p><p>Item 2</p><p>Item 3</p>
23     </div>
24   </div>
25 </body>
26 </html>
```

ch0601.html

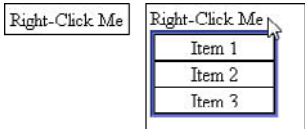


Figure 6.2 Right-clicking on the text pops up a simple list of items.

Handling Keyboard Events

An often forgotten event when working with web pages is the keyboard event. However, keyboard events can be some of the most useful, especially if you are trying to make your web site feel more like a web app.

Users are accustomed to hot keys that perform different functions and interactions as you enter data into fields in applications. Keyboard events let you accomplish just that. The following phrases show you how to detect keyboard input and utilize the event object to determine which key(s) were pressed.

Detecting Changes to Text Input Elements

```
$("input[type=text]").on("keypress",
  function(){
    $("#p1").html("Key Pressed"); });
$("input[type=text]").on("keydown",
  function(){
    $("#p1").html("Key is Down"); });
$("input[type=text]").on("keyup",
  function(){
    $("#p1").html("Key is Up"); });
```

JavaScript provides three events that are triggered when a key is pressed on the keyboard: `keydown`, `keypress`, and `keyup`. These events allow you to capture each keystroke into the text input.

For example, if you want to add an event handler that is triggered each time a key is pressed in text input, you could use the following:

```
$("input[type=text]").on("keypress",
  function(){handler_code});
```

Did you know?

A useful trick when getting the text in a text box while handling keyboard events to input elements is to use the `keyup` instead of the `keypress` event. When the `keypress` event is triggered, the value of the input box may not include the typed key yet. However, if you use the `keyup` event, the textbox value will always include the typed key.

Determining What Key Was Pressed

```
$("input[type=text]").on("keypress",
  function(e){
    var s = "You Pressed ";
    if (e.ctrlKey){ s += "CTRL + "; };
    if (e.altKey){ s += "SHIFT + "; };
    if (e.shiftKey){ s += "SHIFT + "; };
    s += String.fromCharCode(e.charCode);
    $("#p1").html(s);
    });
});
```

The event passed to the event keyboard event handler includes the character code and the auxiliary key information. To determine which key was pressed, use the `event.charCode` attribute to get the character code. You can convert the character code to a character using the `String.fromCharCode(code)` method. For example:

```
var charcter = String.fromCharCode(event.charCode);
```

To determine if the Ctrl, Alt, or Shift keys were pressed, you can check the `event.shiftKey`, `event.altKey`, and `event.ctrlKey` attributes. These are boolean values that indicate whether that key was pressed.

The following code shows an example of using the event data to determine which key(s) were pressed and display that information to the web page. Figure 6.3 shows the resulting web page interaction:

```
<html>
<head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    $(document).ready(function(){
      $("input[type=text]").on("keypress",
        function(e){
          var s = "You Pressed ";
          if (e.ctrlKey){ s += "CTRL + "; };
          if (e.altKey){ s += "SHIFT + "; };
          if (e.shiftKey){ s += "SHIFT + "; };
          s += String.fromCharCode(e.charCode);
          $("#p1").html(s);
        });
    });
  </script>
</head>
<body>
  <p id="p1"></p>
  <input type="text" />
</body>
</html>
```

ch0602.html

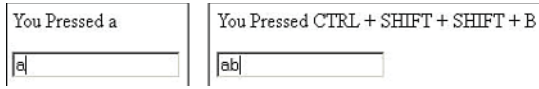


Figure 6.3 Getting the keystroke events in an event handler using JavaScript in ch0602.html.

Form Events

Some of the most important events especially for e-commerce sites are form events. Form events are used heavily when providing users with cart interactions, checkouts, registrations, and more. Adding event handlers to form events allows you to provide some dynamic interactions with the user as the user fills out the forms.

The following phrases are designed to cover the basic concepts of handling the different types of form events. Additionally, there are keyboard and mouse events in the form that have already been covered in this chapter.

Handling Focus Changes

```
$("input").on("focus",
  function(e){
    gain_focus_code_here
});
$("input").on("blur",
  function(e){
    lose_focus_code_here
});
```

When handling form events, two important events are `focus` and `blur`. These events are triggered when a form element receives or loses focus. There is a lot that you can use these events for. For instance, you might want to change the look and feel of a form element when it gains focus or loses focus.

Handling Selection Changes

```
$("select").on("change",
  function(e) {
    select_change_code_here });
$("input[type=checkbox]").on("change",
  function(e) {
    checkbox_change_code_here });
$(" input[type=radio]").on("change",
  function(e) {
    radio_change_code_here });
```

Another important type of event to handle when working with forms is the selection change in check boxes, radio inputs, and select elements. All these elements have a change event handler that is triggered when the selection changes.

To handle selection changes on these items in jQuery, add the following event handler. Then put whatever change code you would like inside the handler function:

```
$("#formElement").on("change",
  function(e) {
    change_code_here
});
```

This page intentionally left blank

7

Manipulating Web Page Elements Dynamically

At the heart of dynamic programming is programmatically altering the existing web page elements on the fly to react to user input. This provides the rich interactive experience that people are beginning to expect from all web sites.

jQuery and JavaScript provide several methods to easily alter the appearance, style, and content of the page elements, from changing an image source and paragraph content to dynamically highlighting text and container borders.

The following sections describe how to utilize jQuery and JavaScript to access and manipulate the web page elements.

Getting and Setting DOM Element Attributes and Properties

DOM objects provide direct access to the DOM object attributes as well as the DOM element properties of the HTML elements they represent. This really is not reasonable in jQuery, because jQuery objects often represent multiple elements with varying attributes. For that reason, jQuery provides the `.attr()` and `.prop()` methods to get and set the attributes and properties of these elements.

The `.attr(attribute, [value])` method allows you to specify an attribute name only to get the current value as well as an optional value to set the current value. For example, the following code gets the `src` value for a specific image element with `id="bannerImg"`:

```
var state = $("#bannerImg").attr("src");
```

Then the following statement sets the `src` attribute value for all `<img>` elements:

```
$("img").attr("src","images/default.jpg");
```

The `.prop(property, [value])` method allows you to specify the property to get the current value and an optional value to set the current value. For example, the following code gets the checked state a specific element with `id="firstCheckbox"`:

```
var state = $("#firstCheckbox").prop("checked");
```

And the following statement sets the `checked` value of all `<input>` elements to `true`:

```
$("input").prop("checked", true);
```

Did you know?

The only difference between a property and an attribute as far as jQuery goes is that attributes are values that define the HTML structure, and properties are values that affect the dynamic state of the object. For example, in an `<input>` element, `"type"` is an attribute because it defines the structure, whereas "`checked`" is a property because it only affects the state.

Changing a Link Location

```
<script>
  $("#pageLink").attr("href", "complexpage.html");
</script
. . .
<body>
  <a id="pageLink" href="simplepage.html">Link</a>
</body>
```

With web pages becoming more and more dynamic, even the links that are built into have become dynamic. A great advantage of using jQuery and JavaScript in your pages is that you can easily alter the URL that links point to dynamically based on user input or other sources of data.

To change the URL that a link points to, use the `.attr("href", newURL)` method. When the user clicks on the link, the browser reads the `href` attribute of the `<a>` element and then loads that location as the next page.

Changing an Image Source File

```
var imgArr = ["bison.jpg","peak.jpg","falls.jpg"];
var idx = 0;
$(document).ready(function (){
  $("img").on("click", function(){
    if(idx<2) { idx++; } else { idx=0; }
    $(this).attr("src", imgArr[idx]);
  });
});
```

A common task in jQuery and JavaScript is changing the images displayed on the screen. Whether it is an online ad, a game, or an image gallery, it is much nicer to simply swap out the source file for an `<img>` element than reload the web page.

To change the image that is displayed in an `<img>` element, use the `.attr("src", newImageURL)` method to specify a new location of the new user file. When the `src` attribute changes, the browser automatically downloads the new image if needed and renders it in the web page without requiring a reload.

The following code shows a good example of swapping out images, as illustrated in Figure 7.1. A single image is in the web page. On line 13, in a `click` handler, the `src` attribute is changed to one of the values from the `imgArr` array:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     var imgArr =
```

```
➥["bison.jpg","peak.jpg","falls.jpg"];
09      var idx = 0;
10      $(document).ready(function (){
11        $("img").on("click", function(){
12          if(idx<2) { idx++; } else { idx=0; }
13          $(this).attr("src", imgArr[idx]);
14        });
15      });
16    </script>
17    <style>
18      img { width:300px; border:3px ridge white;
19         box-shadow: 5px 5px 5px #888888;
➥margin:10px; }
20    </style>
21  </head>
22    <body>
23      <img src="bison.jpg" />
24    </body>
25  </html>
```

ch0701.html

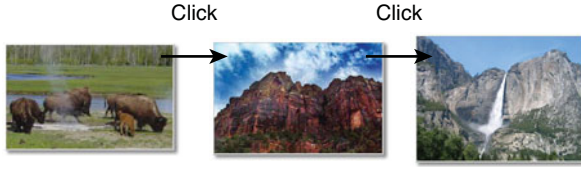


Figure 7.1 Swapping images using jQuery code in ch0701.html.

Getting and Setting CSS Properties

jQuery makes it extremely easy to get and set CSS values using the `.css(property, [value])` method. For example, the following code retrieves the `cursor` CSS property value of an element:

```
$("#buttonA").css("cursor");
```

Then the following sets the `border-radius` value:

```
$("#buttonA").css("border-radius", "10px 15px");
```

The `.css()` method also allows you to pass a map object with properties and values. This allows you to set several settings at once. For example, the following code uses `.css()` to set the `margin`, `padding`, `float`, and `font-weight` attributes at the same time:

```
$("span").css({margin:0, padding:2, float:"left",
➥"font-weight":"bold"});
```

Notice that the property names can either be enclosed in quotes or not. You need to use quotes if the property name contains a - or another character that is not valid in a JavaScript object name. The values can be numbers or strings. The numerical values represent distance, which can be expressed in `px`, `cm`, or `%`. The default is `px`, so if you want to specify pixels, you only need to enter the number. If you want to specify `cm`, `%`, or some other value type, you need to use a string, such as `"100%"`.

Changing Colors

```
$("h1").on("click", function(){
  $(this).css({color:"white",
               "background-color":"black"});
});
```

A great way to provide interaction with user actions is to alter the colors of elements on the screen. Colors can provide additional meaning. For instance, changing the color of text to red or the background color of text to a yellow highlight accentuates the text as important to read.

In jQuery, you can change all the CSS attributes that control the colors on the screen using the `.css(property, value)` method. However, if you are changing multiple color properties, it makes much more sense to use the `.css({property:value, property:value...})` method, which allows you to change several properties at once.

The following code provides a good example of using jQuery to dynamically change the color of a header when it is clicked on. A `click` handler alters the background and foreground colors so that the header stands out more, as illustrated in Figure 7.2. Notice that the `background-property` key had to be placed in quotes to support the JavaScript language syntax.

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
```

```
08      $(document).ready(function (){
09        $("h1").on("click", function(){
10          $(this).css({color:"white",
11                       "background-
➥color":"black"});
12        });
13      });
14    </script>
15  </head>
16    <body>
17      <h1>Heading 1</h1>
18      <h1>Heading 2</h1>
19    </body>
20  </html>
```

ch0702.html

Figure 7.2 Changing foreground and background colors using jQuery code in ch0702.html.

Adding Borders

```
$("ul").on("click", function(){
  $(this).css({"border":"3px groove black"});
});
$("li").on("click", function(){
  $(this).css({"border-style":"dotted",
               "border-size":1,
               "border-color":"blue"});
});
```

Another useful style that can be programmatically applied to elements is borders. Borders give you a way to highlight specific elements on the page or just change the look of things.

You can change the CSS attributes that apply borders to elements in jQuery using the `.css(property, value)` or `.css({property:value, property:value...})` method.

By the way

You can set all the border options at the same time using the border property with a size, style, and color, for example:

```
$(this).css({"border":"3px groove black"});
```

The following code provides a good example of using jQuery to dynamically add borders to `<ul>` and `<li>` elements. I provide two examples: one that changes all attributes in a single border property setting, and the other that changes them individually. A `click` handler applies the borders to the list element and list item elements illustrated in Figure 7.3:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function (){
09       $("ul").on("click", function(){
10         $(this).css({"border":"3px groove
➥black"});
```

```
      });
      $("li").on("click", function(){
        $(this).css({"border-style":"dotted",
                     "border-size":1,
                     "border-color":"blue"});
      });
    });
  </script>
</head>
  <body>
    <ul>
      <li>George Washington</li>
      <li>John Adams</li>
      <li>Thomas Jefferson</li>
    </ul>
  </body>
</html>
```

ch0703.html

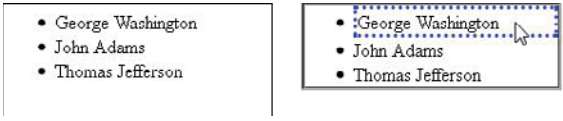


Figure 7.3 Adding borders using jQuery code in ch0703.html.

Changing Fonts

```
$("p").on("click", function(){
  $(this).css({"font-weight":"bold",
       "font-style":"italic",
       "font-size":22,
       "font-family":"cursive, sans-serif"});
});
```

A fun option available using the `.css()` method is the ability to alter the font of paragraphs and other elements. You can use the `.css()` method for a variety of purposes, such as illustrating an area already read or accentuating text that becomes contextually important. You can adjust the `font-weight`, `font-size`, `font-style`, and `font-family` attributes directly to change the appearance of the text.

To programmatically alter the font values, apply the `.css(property, value)` or `.css({property:value, property:value...})` method. The following code provides a good example of using jQuery to dynamically change the style of a paragraph once you click it. The `click` handler, shown in lines 9–14, applies the font changes when the user clicks on the paragraph, as illustrated in Figure 7.4:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function (){
09       $("p").on("click", function(){
10         $(this).css({"font-weight":"bold",
11              "font-style":"italic",
12              "font-size":22,
13              "font-family":"cursive, sans-
➥serif"});
14       });
15     });
16   </script>
17 </head>
```

```
18   <body>
19     <p>The play's the thing, wherein
20        I'll catch the conscience of the king.</p>
21   </body>
22 </html>
```

ch0704.html

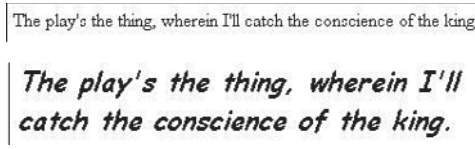


Figure 7.4 Changing font styles using jQuery code in ch0704.html.

Adding a Class

```
$("span").addClass("baseClass");
```

An important part of rich interactive web pages is good CSS design. JavaScript and jQuery can enhance the CSS design by dynamically adding and removing classes from elements.

jQuery makes it extremely simple to add, remove, and toggle classes on and off. If you design your CSS code well, it's easy to apply some nice effects.

You add classes using the `.addClass(className)` method. For example, to add a class named `active` to all `<span>` elements, you could use the following statement:

```
$("span").addClass("active");
```

Removing a Class

```
$("span").removeClass("oldClass");
```

You remove classes using the `.removeClass([className])` method. For example, to remove the active class from the `<span>` elements, you call the following:

```
$("span").removeClass("active");
```

You can also use `remove` with no `className`, which removes all classes from the elements. For example, the following statement removes all classes from `<p>` elements:

```
$("p").removeClass();
```

Toggling Classes

```
$("span").on("click", function(){
   $(this).toggleClass("active");
});
```

You can toggle classes on and off using the `.toggleClass(className [, switch)` method. In addition to the `className`, you can specify `true` or `false` for the optional `switch` parameter, indicating to turn the class on or off.

For example, to turn the active class and the inactive class off for all `<span>` elements, the code would be as follows:

```
$("span").toggleClass("active", true);
$("span").toggleClass("inactive", false);
```

The following code provides a good example of using jQuery to dynamically toggle a class on and off when a user clicks on a `<span>` element. The `click` handler in lines 9–11 uses the `.toggleClass()` method to switch the `.active` class on and off. Figure 7.5 illustrates the look and behavior of the code:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function (){
09       $("span").on("click", function(){
10         $(this).toggleClass("active");
11       });
12     });
13   </script>
14   <style>
15     span {
16       border:3px ridge white; border-radius:10px;
17       padding:3px; color:black; background-
➥color:#CCCCCC;
18       cursor:pointer; }
19     .active {
20       background-color:blue; color:white;
21       border-color:blue; font-weight:bold;}
22   </style>
23 </head>
24   <body>
25     <span>Option 1</span>
26     <span>Option 2</span>
27   </body>
28 </html>
```

ch0705.html

Figure 7.5 Toggling classes on and off using jQuery code in ch0705.html.

Getting and Manipulating Element Content

Often, you will not know all the elements that belong on a web page until a user begins using it, or you will receive additional information from a web service or some other interaction. In those cases, you may need to be able to add or replace element content on the fly using jQuery and JavaScript code.

The text of the element content is stored in the `innerHTML` attribute of the DOM element. You can change the content using JavaScript simply by setting the `innerHTML` to the new content. For example:

```
document.getElementById("#p1").innerHTML = "New
➥Text";
```

In jQuery, you can change the `innerHTML` content using the `.html()` method, which gets or sets the `innerHTML` sting. For example:

```
$("#p1").html("Paragraph 1 goes here");
```

Getting the Content of an HTML Element

```
var p1Contents = $("#p1").html();
var div1Contents = $("#div1").html();
```

Containing the content inside an HTML element is easy using the jQuery `.html()` method. Calling this method with no parameter returns an HTML string version of the content. If the content is just text, the return is a simple string. If the content is HTML elements, the string is included in the HTML tags.

Appending and Prepending Text to a Basic HTML Element

```
$("#p1").prepend("*New* :");
$("#p1").append(". . .read");
```

Another common dynamic task is adding text to a basic HTML element such as a span or paragraph. jQuery makes this extremely simple using the `.append()` method. This method appends content to the end of the element's current content. It doesn't matter if the content is a string or other elements.

For example, the following code appends the text `". . .read"` to a paragraph with `id="p1"`:

```
$("#p1").append(". . .read");
```

You may also want to prepend text to the beginning of an element's content. In jQuery, the `.prepend()` method prepends content to the beginning of the element's current content. It doesn't matter if the content is a string or other elements.

For example, the following code prepends the text `"*New* :"` to a paragraph with `id="p1"`:

```
$("#p1").prepend("*New* :");
```

Replacing Element Text

```
$("$mySpan").html("content");
. . .
$("$mySpan").html("newer content");
. . .
$("$mySpan").html("even newer content");
```

Rather than prepending or appending text (or content, for that matter) to an element, you may want to replace the element content entirely. The simplest way to do this in jQuery is to use `.html()`, which sets the value of the content to whatever string or object you pass into it, thus replacing the existing content.

Appending Elements to Parent Content

```
var newP = $("<p></p>");
newP.html("This is the first paragraph.");
$("div").prepend(newP);
newP.html("This is the last paragraph.");
$("div").append(newP);
```

You can also use the `.prepend()` and `.append()` methods in jQuery to prepend or append content to the element. Another major advantage of jQuery is that you can append HTML DOM elements created using the jQuery constructor `jquery(htmlString)` or `$(htmlString)` to add new HTML elements to the page. For example, the following code constructs a jQuery object with a `<p>` element, sets the content of the new paragraph element using `.html()`, and then appends the paragraph to all `<div>` elements.

```
var newP = $("<p></p>");
newP.html("This is a new paragraph.");
$("div").append(newP);
```

This page intentionally left blank

8

Manipulating Web Page Layout Dynamically

One of the coolest interactions that you can make with web pages is to rearrange elements on the page based on user interaction. For instance, you can make elements bigger or smaller and even change the position. Additionally, you can change the visibility of elements, hiding them when you don't need them and revealing them when appropriate.

This chapter focuses on the methods to manipulate elements in such a way that it alters the basic layout of the page. The phrases are designed to give you usable examples that apply to an array of purposes.

Hiding and Showing Elements

```
$("#down").hide();
$("#up").on("click", function(){
  $("#up, #down").toggle();
  $("#leftNav").hide(); });
$("#down").on("click", function(){
  $("#up, #down").toggle();
  $("#leftNav").show(); });
```

A simple way of changing the look and feel of web pages is to toggle the visibility of elements. Hiding elements that are not necessary and then only showing them when they become necessary can save a lot of screen space that can be critical in a well-implemented web application.

You hide or show elements from JavaScript by setting the `style.display` property to `"none"` or to `""`. jQuery, in contrast, provides a much more elegant and extensible solution.

To display an element using jQuery, simply call the `.show()` method on the jQuery object. All items in the object set are shown. Then to hide the elements, use the `.hide()` method. It's as simple as that. For example, to hide all `<p>` elements, you would use the following:

```
$("p").hide();
```

Then to display them again, use this:

```
j$("p").show();
```

Another valuable tool that jQuery provides is the `.toggle()` method. This method changes the current visibility state to the exact opposite. That means you don't have to keep track or check to see what the visibility is for items that you simply want to toggle on and off.

The following code demonstrates changing the visibility. It hides and shows a simple popup menu. Notice that the buttons to show and hide the visibility are themselves being toggled on and off so that the appropriate button is visible to match the menu state. Figure 8.1 shows the basic look and feel of the menu:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-1.9.0.min.js"></script>
  <script>
    var imgArr =
➥["bison.jpg","peak.jpg","falls.jpg"];
    var idx = 0;
    $(document).ready(function (){
      $("#down").hide();
      $("#up").on("click", function(){
        $("#up, #down").toggle();
        $("#leftNav").hide(); });
      $("#down").on("click", function(){
        $("#up, #down").toggle();
        $("#leftNav").show(); });
    });
  </script>
  <style>
    #banner { font:bold 36px/60px "cursive,
➥serif";
      color:white; background-color:blue;
      text-align:center; }
    #bar { background-color:black; padding:3px;}
    #leftNav {background-color:#cccccc;
➥padding:5px;
      width:100px; }
    p { margin:0; margin-top:2px; border:1px
➥solid;
        text-align:center; border-radius:10px;
        background-color:white; }
  </style>
</head>
  <body>
    <div id="banner">jQuery and JavaScript</div>
```

```
34      <div id="bar"><img id="down" src="expand.png"
➥/>
35        <img id="up" src="collapse.png" /></div>
36      <div id="leftNav"><p>Option 1</p>
37        <p>Option 2</p><p>Option 3</p></div>
38    </body>
39 </html>
```

ch0801.html

Figure 8.1 Toggling the visibility of a menu item when it is not needed using jQuery `.show()`, `.hide()`, and `.toggle()`.

Adjusting Opacity

```
$("#up").on("click", function(){
  var op = parseFloat($("#photo").css("opacity"));
  if (op<1) {op += 0.2;};
  $("#photo").css("opacity", op);
});
$("#down").on("click", function(){
  var op = $("#photo").css("opacity");
  if (op>0) {op -= 0.2;};
  $("#photo").css("opacity", op);
});
```

Changing the opacity of elements is a great way to alter the layout of the page. Lowering the opacity value makes the item less visible, revealing any elements or background images that are behind the item.

By the way

One problem with using `.hide()` to hide an element is that, once applied, the element no longer takes up page space. Other flowing elements slide in to take its place. In many cases, this is the way you want it. In other instances, you may want the element to be invisible but still take up space, so set opacity to 0.

Lowering the opacity (but not to 0) can be a great way to demonstrate that elements are not currently active while still showing them. For example, I like to set menu and button elements that are not yet implemented and active to .5 opacity so that they still show up but are obviously not clickable.

The opacity CSS property controls the opacity. To make an element invisible but still take up space, set the opacity CSS property to 0. For example:

```
jObj.css("opacity", "0");
```

Then to make the element visible again, set the opacity back to 1:

```
jObj.css("opacity", "1");
```

To make an item partially visible, set the opacity between 0 and 1, such as .5. The following example creates a web page with two buttons that control the opacity of an image element. As the opacity increases, the image becomes more visible and vice versa. The click handlers for the buttons change the opacity by incrementing or decrementing the current opacity by .2. Figure 8.2 shows the opacity changes in action:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-1.9.0.min.js"></script>
  <script>
    var imgArr =
➥["bison.jpg","peak.jpg","falls.jpg"];
    var idx = 0;
    $(document).ready(function (){
      $("#up").on("click", function(){
        var op =
➥parseFloat($("#photo").css("opacity"));
        if (op<1) {op += 0.2;};
        $("#photo").css("opacity", op);
      });
      $("#down").on("click", function(){
        var op = $("#photo").css("opacity");
        if (op>0) {op -= 0.2;};
        $("#photo").css("opacity", op);
      });
    });
  </script>
  <style>
    img{ vertical-align:middle;}
    #photo { width:300px; border:5px ridge white;
       box-shadow: 5px 5px 5px #888888;
➥margin:10px; }
  </style>
</head>
  <body>
    <img id="down" src="decrease.png" />
    <img id="photo" src="arch.jpg" />
    <img id="up" src="increase.png" />
```

```
33   </body>
34 </html>
```

ch0802.html

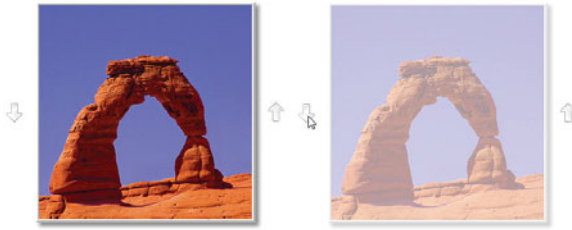


Figure 8.2 Toggling the opacity of an image using the jQuery code in ch0802.html.

Resizing Elements

```
$("#up").on("click", function(){
  $("#photo").width($("#photo").width()*1.2);
});
$("#down").on("click", function(){
  $("#photo").width($("#photo").width()*.8);
});
```

Another important aspect when dynamically working with HTML elements is the ability to get and change an element's size. jQuery makes this simple using a set of methods attached to the jQuery object. Table 8.1 shows the methods provided by jQuery objects that allow you to get the height and width of an element.

Table 8.1 **jQuery Object Methods to Get and Set the Element Size**

Attribute	Description
`height([value])`	If a value is specified, the height of all the HTML elements in the set is changed; otherwise, the current height of the first HTML element is returned.
`width([value])`	If a value is specified, the width of all the HTML elements in the set is changed; otherwise, the current width of the first HTML element is returned.
`innerHeight()`	Returns the current height, including padding, of the first element in the set.
`innerWidth`	Returns the current width, including padding, of the first element in the set.
`outerHeight ([includeMargin])`	Returns the current height, including padding, border, and margin if specified, of the first element in the set.
`outerWidth ([includeMargin])`	Returns the current width, including padding, border, and margin if specified, of the first element in the set.

Watch out!

The height and width methods return only the size of the first element in the jQuery objects' set. For single object sets, that is not a problem. Just keep in mind that other objects in the set may have different sizes.

To illustrate getting and setting the size of elements, consider the following sample code. This code defines a web page with a photo and two buttons. The click handlers for the buttons increase or decrease the size of the image by getting the image width using the `.width()` method. The image automatically resizes as the width changes in the handlers. To adjust the size, pass back a new value of `width*1.2` or `width*.8` into the `.width()` method. Figure 8.3 shows the buttons that resize the image:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-1.9.0.min.js"></script>
  <script>
    $(document).ready(function (){
      $("#up").on("click", function(){

➥$("#photo").width($("#photo").width()*1.2);
      });
      $("#down").on("click", function(){

➥$("#photo").width($("#photo").width()*.8);
      });
    });
  </script>
  <style>
    img{ vertical-align:middle;}
    #photo { width:300px; border:5px ridge white;
       box-shadow: 5px 5px 5px #888888;
➥margin:10px; }
  </style>
</head>
```

```
23   <body>
24     <img id="down" src="decrease.png" />
25     <img id="photo" src="arch.jpg" />
26     <img id="up" src="increase.png" />
27   </body>
28 </html>
```

ch0803.html

Figure 8.3 Resizing an image using the jQuery code in ch0803.html.

Repositioning Elements

```
$("#up").on("click", function(){
  var offset = $("#photo").offset();
  offset.top -= 10;
  $("#photo").offset(offset);
});
```

In addition to the size of HTML elements, you often need to determine their position. There are two different types of positions when working with HTML elements. The first type is the position relative to the full document. The second type is the position relative to

the HTML element that acts as an offset parent. The element that is the offset parent depends on the position settings in CSS.

jQuery provides the `.position([position])` method to get the position relative to the offset parent. The `.offset([position])` method provides the position relative to the document. You can call both of these methods with no argument and return a simple object with a left and right attribute that represent the number of pixels from the left and top of document or offset parent. You can also call these methods with a simple `position` object with `left` and `right` attributes, in which case they set the position of the element.

For example, the following code retrieves the number of pixels from the top and the left of the document as well as the number of pixels from the top and left of the offset parent for the element with `id="myElement"`. To get each value using a single statement I reference the `top` and `left` attributes directly after the `offset` of `position` call:

```
var pixelsFromPageTop =
➥$("#myElement").offset().top;
var pixelsFromPageLeft = $("#myElement").offset
➥().left;
var pixelsFromParentTop = $("#myElement").position
➥().top;
var pixelsFromParentLeft = $("#myElement").position
➥().left;
```

To set the distance of that element exactly 10 pixels down and 10 pixels to the right of the top-left corner of the document, use the following statement, which defines a simple object with `left` and `top` values and passes it to the `.offset()` method:

```
$("#myElement").offset({"top":10,"left":10);
```

To illustrate using element position, the following code builds a web page with four control buttons and an image element. The click handlers for the buttons move the image around by getting the current offset, adjusting the top or left attribute, and then setting the offset again using the `.offset()` method. Figure 8.4 shows the buttons moving the image around the screen:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-1.9.0.min.js"></script>
  <script>
    var imgArr =
➥["bison.jpg","peak.jpg","falls.jpg"];
    var idx = 0;
    $(document).ready(function (){
      $("#up").on("click", function(){
        var offset = $("#photo").offset();
        offset.top -= 10;
        $("#photo").offset(offset);
      });
      $("#down").on("click", function(){
        var offset = $("#photo").offset();
        offset.top += 10;
        $("#photo").offset(offset);
      });
      $("#left").on("click", function(){
        var offset = $("#photo").offset();
        offset.left -= 10;
        $("#photo").offset(offset);
```

```
25        });
26        $("#right").on("click", function(){
27          var offset = $("#photo").offset();
28          offset.left += 10;
29          $("#photo").offset(offset);
30        });
31      });
32    </script>
33    <style>
34      img{ vertical-align:middle;}
35      #photo { width:300px; border:5px ridge white;
36         box-shadow: 5px 5px 5px #888888; position:
➥fixed; }
37    </style>
38  </head>
39    <body>
40      <div id="buttons">
41        <img id="up" src="up.png" />
42        <img id="down" src="down.png" />
43        <img id="left" src="left.png" />
44        <img id="right" src="right.png" />
45      </div>
46      <img id="photo" src="arch.jpg" />
47    </body>
48  </html>
```

ch0804.html

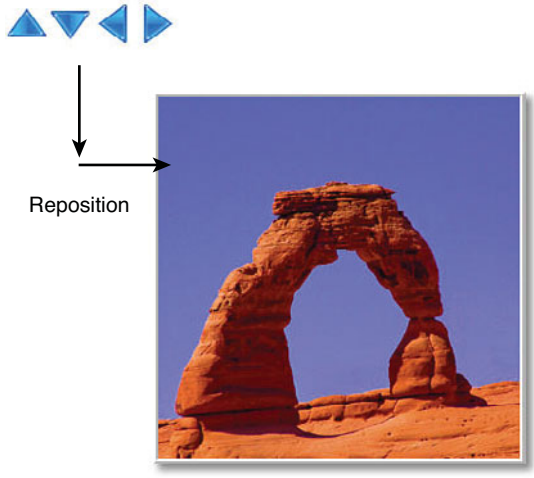


Figure 8.4 Moving an image element around the screen using the jQuery code in ch0804.html.

Stacking Elements

```
$("img").on("click", function(){
  $("img").css("z-index", 0);
  $(this).css("z-index", 1);
});
```

One of the coolest interactions that you can make with web pages is to rearrange elements on top of each other based on user interaction. The idea is that HTML elements can simply be stacked on top of each other like papers on a desk. At any time, you can reveal one of the hidden or partially hidden elements and place it on top of the others.

The z-index is a CSS property that specifies the position of an HTML element with respect to other elements not vertically or horizontally, but projected out toward the user. The element with the highest z-index is displayed on top of other elements when the browser renders the page.

To get and set the z-index in jQuery, use the .css() method. For example, to get the z-index for an item, use this:

```
var zIndex = $("#item").css("z-index");
```

To set the z-index for an item to, say, 10, use the following statement:

```
$("#item").css("z-index", "10");
```

To illustrate how the z-index works, the following code generates a web page with three images. The images have fixed positioning in an overlapping fashion. When you click on one of the images, the click handler first adjusts the z-index of all items to 0 and then sets the z-index for the item clicked on to 1, thus placing it on top. This is shown in Figure 8.5:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-1.9.0.min.js"></script>
07   <script>
08     var imgArr =
➥["bison.jpg","peak.jpg","falls.jpg"];
09     var idx = 0;
```

```
10      $(document).ready(function (){
11        $("img").on("click", function(){
12          $("img").css("z-index", 0);
13          $(this).css("z-index", 1);
14        });
15      });
16    </script>
17    <style>
18      img{ width:300px; border:5px ridge white;
19          box-shadow: 5px 5px 5px #888888; position:
➥fixed; }
20      #photo1 { top:10px; left:20px; }
21      #photo2 { top:60px; left:60px; }
22      #photo3 { top:120px; left:120px; }
23    </style>
24 </head>
25    <body>
26      <img id="photo1" src="arch.jpg" />
27      <img id="photo2" src="beach.jpg" />
28      <img id="photo3" src="flower.jpg" />
29    </body>
30 </html>
```

ch0805.html

Figure 8.5 Stacking elements on top of each other and reordering them using the jQuery code in ch0805.html.

9

Dynamically Working with Form Elements

Web forms are an integral part of dynamic web programming. You may think of forms only in terms of credit card payments, online registration, and the like, but anytime you need actual data input from the user, you are using web forms.

Web forms can be a positive or negative experience for users. If it seems difficult to input the data on the form because the form is clunky or difficult to understand, users hate it and have a bad experience.

Applying dynamic adjustment to the web form by dynamically accessing and manipulating the flow of the form allows you to give users a much better experience. The phrases in this chapter focus on accessing the form data and then using that data to dynamically manipulate the form to improve the overall user experience.

Getting and Setting Text Input Values

```
var inText = $("#inBox").val();
if (inText != "default"){
  $("#outBox").val(inText);
}
```

The most common type of form elements are textual inputs. These elements include the `<textarea>` element as well as `<input>` elements with the following `type` attribute values: `color`, `date`, `datetime`, `datetime-local`, `email`, `month`, `number`, `password`, `range`, `search`, `tel`, `text`, `time`, `url`, and `week`.

Although the browser uses these values a bit differently, all are rendered in the same basic text box and are accessed in the same basic way. Each of them has a value attribute that is displayed in the text box as the image is rendered.

In jQuery, you can access the value text inputs using the `.val()` method of the jQuery object. To get the text typed into the text box, you should call the `.val()` method with no arguments. For example, to get the value of `#textbox1`, you would use the following:

```
var textbox1text = $("#textbox1").val();
```

To set the text that appears in the text box, you would use `.val(string)`. For example, to set the value of `#textbox1` to "`new string`", you would use this:

```
$("#textbox1").val("new string");
```

Watch out!

The `.attr()`, `.prop()`, and `.val()` methods only get the values of the first element in the matched jQuery set. If you are working with multiple elements in the set, you may need to use a `.map()` or `.each()` method to get values from all elements.

Checking and Changing Check Box State

```
if ($("#myCheckbox").is(":checked)){
  $("#myCheckbox").removeAttr("checked");
else{
  $("#myCheckbox").attr("checked");
}
```

Check box input elements have a boolean value based on whether the element is checked. You access the value by getting the value of the `checked` attribute. If the element is checked, then `checked` has a value such as `true` or "`checked`". Otherwise, the value will be `undefined` or `false`.

You can get and set the state of a check box element from JavaScript in the following manner:

```
domObj.checked = true;
domObj.checked = false;
var state = domObj.checked;
```

Determining whether an item is checked with jQuery is a bit different. Remember, in jQuery you may be dealing with multiple check boxes at once, so the safest way to see if the jQuery object represents an object that is checked is the `.is()` method. For example:

```
$("#myCheckbox").is(":checked");
```

To set the state of a jQuery object representing check boxes to checked, you would simply set the `checked` attribute as follows:

```
$("#myCheckbox").attr("checked", true);
```

Setting the state of a jQuery object representing check boxes to unchecked is a bit different. You need to remove the `checked` attribute using `removeAttr()`. For example:

```
$("#myCheckbox").removeAttr("checked");
```

Getting and Setting the Selected Option in a Radio Group

```
if ($("#maleRB").is(":checked")){
  $("#maleRB").removeAttr("checked");
else{
  $("#maleRB").attr("checked");
}
var genderGroup = $("input[name=gender]");
var checkedGender = genderGroup.filter(":checked");
var selectedGender = checkedGender.val();
```

Radio inputs are almost always used in groups. The value of a radio input that a group represents is not boolean. Instead, it is the `value` attribute of the currently selected element. For example, the value of the following radio button group is either `"male"` or `"female"`:

```
<input id="maleRB" type="radio" name="gender"
➥value="male">
<label for="maleRB">Male</label>
<input id="femaleRB" type="radio" name="gender"
➥value="female">
<label for="femaleRB">Female</label>
```

To get the value of a radio input group in code, you first need to access all the elements in the group, find out which one is selected, and then get the `value` attribute from that object. The following code gets the value of a radio input group that is grouped by `name="gender"` in jQuery:

```
var genderGroup = $("input[name=gender]");
var checkedGender = genderGroup.filter(":checked");
var selectedGender = checkedGender.val();
```

Setting the checked value of individual radio inputs works the same way as check boxes. To see if the jQuery object represents a radio button that is checked, use the `.is()` method. For example:

```
$("#maleRB").is(":checked");
```

To set the state of a jQuery object representing a radio button to checked, you simply set the `checked` attribute as follows:

```
$("#maleRB").attr("checked", true);
```

To set the state of a jQuery object representing a radio button to unchecked, you need to remove the `checked` attribute using `removeAttr()`. For example:

```
$("#maleRB").removeAttr("checked");
```

Getting and Setting Select Values

```
var value = $("#singleSelect").val();
var values = $("#multiSelect").val();
. . .
$("#singleSelect").val("one");
$("#multiSelect").val(["two", "three"]);
```

Select inputs are really container inputs for a series of `<option>` elements. The value of the select element is the value(s) of the currently selected option(s). Once again, the submission and serialization in jQuery and JavaScript handle this automatically for you. However, doing it manually requires a bit of code.

Did you know?

If you do not specify a value attribute for an `<option>` element, the value returned is the value of the innerHTML. For example, the value of the following option is `"one"`:

```
<option>one</option>
```

You may want a couple of different values when accessing a `<select>` element. One is the full value represented by the element. Getting that value is simple in jQuery using the `.val()` method. For example, consider the following code:

HTML:

```
<select id="mySelect">
  <option value="one">One</option>
  <option value="two">Two</option>
  <option value="three">Three</option>
</select>
```

jQuery:

```
$("#mySelect").val();
```

The value returned by the jQuery statement if the first option is selected is this:

```
"one"
```

For multiple selects, the `.val()` method returns an array of the values instead of a single value. On a multiple select, the value returned by the jQuery statement if the first option is selected is this:

```
["one"]
```

On a multiple select, the value returned by the jQuery statement if the first option is selected follows:

```
["one", "two", "three"]
```

You can also use the `.val()` method to set the selected elements. For example, the following statement selects the second element in the previous select:

```
$("#mySelect").val("one");
```

The following statement selects the second and third options in a multiple select element:

```
$("#mySelect").val(["two", "three"]);
```

Getting and Setting Hidden Form Attributes

```
var hiddenValue = $("#hidden").val();
var hiddenValue = $("#hidden").attr("other");
var hiddenValue = $("#hidden").prop("otherProp");
var hiddenValue = $("#hidden").data("storedValue");
```

A great HTML element to use if you need to supply additional information to the browser from a form is the hidden input. The hidden input is not displayed with the form, but it can contain a name and value pair that is submitted or even just values that you want to store in the form and have accessible during dynamic operations.

The parts to be sent with the form are the `name` and `value` attributes. However, you can attach additional values to a hidden form object or any HTML DOM object from jQuery using the `.data(key [,value])` method. This method works like `.attr()` and `.prop()` in that you pass it a key if you want to get the value or, alternatively, you pass a key and value if you want to set the value of a key. For example, the following code defines a simple hidden element and then uses jQuery to assign the submission value and an extended attribute:

HTML:

```
<input id="invisibleMan" name="InvisibleMan"
➥type="hidden" />
```

jQuery:

```
$("#invisibleMan").val("alive");
$("#invisibleMan").data("hairColor", "clear");
var state = $("#invisibleMan").val();
var state = $("#invisibleMan").data("hairColor");
```

Disabling Form Elements

```
$("#cbSame").on("click", function(){
  if($(this).is(":checked")){
    $("#billInfo :input").attr("disabled",true); }
  else {
    $("#billInfo :input").removeAttr("disabled"); }
});
```

Disabling web elements still displays them, but it prevents the user from interacting with them. Typically, it only makes sense to disable a form element instead of disable it if you still want the user to be able to see the values of the elements.

To disable a form element, you need to set the `disabled` attribute. In JavaScript, you can do this directly on the DOM object. In jQuery, you use the `.attr()` method. For example:

```
$("#deadElement").attr("disabled", "disabled");
```

To re-enable a disabled element, you remove the `disabled` attribute. For example:

```
$("#deadElement").removeAttr("disabled");
```

The following code shows an example of using a check box to enable and disable multiple form elements. The click handler checks the state of a check

box and, if checked, the input elements inside the `#billInfo <div>` are disabled. Otherwise, they are enabled. Figure 9.1 shows the form elements that are disabled and enabled.

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function (){
09       $("#cbSame").on("click", function(){
10         if($(this).is(":checked")){
11           $("#billInfo
➥:input").attr("disabled",true); }
12         else {
13           $("#billInfo :input").removeAttr
➥("disabled"); }
14       });
15     });
16   </script>
17   <style>
18   </style>
19 </head>
20   <body>
21       <form>
22         <span>Shipping Info</span><br>
23         <div id="shipInfo">
24           <input type="text">Name</input><br>
25           <input type="text">Address</input><br>
26           <input type="text">City</input><br>
27           <select class="state"
➥id="state"></select>
28           <input type="text">Zip</input><br>
29         </div>
```

```
30          <span>Billing Info</span><br>
31          <input type="checkbox" id="cbSame"/>
32              <label for="cbSame">Same as
➥Shipping</label>
33          <div id="billInfo">
34            <input type="text">Name on
➥Card</input><br>
35            <input type="text">Address</input><br>
36            <input type="text">City</input><br>
37            <select class="state"
➥id="state"></select>
38            <input type="text">Zip</input><br>
39          </div>
40        </form>
41    </body>
42 </html>
```

ch0901.html

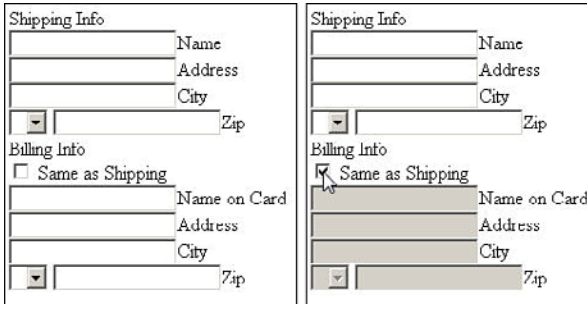


Figure 9.1 Disabling form elements by checking the state of a check box using jQuery code in ch0901.html.

Showing/Hiding Form Elements

```
$("#tires,#model").hide();
$("#make").on("change", function(){
  if($(this).val() == ""){
    $("#tires,#model").hide(); }
  else{ $("#model").show(); }
});
```

Another great trick when providing flow control for a web form is to dynamically hide and show elements. Less is more; you shouldn't necessarily show users more elements than they need to fill out.

For example, if the form has elements for both men's sizes and women's sizes, don't show both. Wait for users to select the gender and then display the appropriate size elements.

Form elements can be shown and hidden in jQuery using the `.show()` and `.hide()` methods. Alternatively, if you want to make the element invisible but still take up space, you can set the opacity CSS attribute to 0 or 1.

The following code shows a basic web form with three selects. Only the select that allows users to select a car make is displayed at first, but as users select the make, a model menu appears. Likewise, as users select the model, a tires size menu appears. The click handlers use `.show()` and `.hide()` to show and hide the menus appropriately. Figure 9.2 shows the form elements being displayed as more items are selected:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
```

```
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     $(document).ready(function (){
09       $("#tires,#model").hide();
10       $("#make").on("change", function(){
11         if($(this).val() == ""){
12           $("#tires,#model").hide(); }
13         else{ $("#model").show(); }
14       });
15       $("#modelS").on("change", function(){
16         var v = $(this).val();
17         if($(this).val() == ""){
18           $("#tires").hide(); }
19         else{
20           $("#tires").show(); }
21       });
22     });
23   </script>
24   <style>
25     select {width:100px; }
26   </style>
27 </head>
28   <body>
29       <form>
30         <select id="make">Maker
31           <option></option>
32           <option>Jeep</option>
33         </select><label>Make</label><br>
34         <div id="model"><select id="modelS">
35           <option></option>
36           <option>Wrangler</option>
37           <option>Cherokee</option>
38         </select><label>Model</label></div>
39         <div id="tires"><select>
40           <option></option>
41           <option>32"</option>
```

```
42            <option>33"</option>
43            <option>36"</option>
44          </select><label>Tires</label></div>
45        </form>
46    </body>
47 </html>
```

ch0902.html

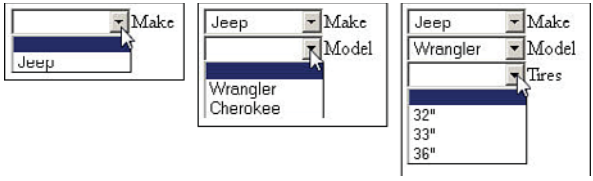


Figure 9.2 Dynamically showing and hiding selects elements using jQuery code in ch0902.html.

Forcing Focus to and Away from Form Elements

```
$("input[type=radio]").on("change",
  function(){
    $("#cardNum").focus();
});
```

A great flow control feature in web forms is to automatically focus elements when you know the user is ready to enter them. For example, if the user selects a year and the next element is a month selection, it makes sense to make the month active for the user automatically.

To set the focus of an element in jQuery, call the `.focus()` method on that object. For example, the following code sets the focus for an object with `id="nextInput"`:

```
$("#nextInput").focus();
```

You can also blur an element that you want to navigate the user away from by calling the `.blur()` method:

```
$("#nextInput").blur();
```

The following code shows an example of a credit card payment form. The logical step for the user to take after selecting the card type is to enter the credit card number. The form detects a change on the radio button group and then immediately shifts focus to the card number input element. Figure 9.3 illustrates the selection automation in the form:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    $(document).ready(function (){
      $("input[type=radio]").on("change",
➥function(){
        $("#cardNum").focus();
      });
    });
  </script>
  <style>
```

```
15   </style>
16 </head>
17   <body>
18   <form>
19     <input type="radio" name="ptype" id="visa" />
20       <label for="visa"><img src="visa.png"
➥/></label>
21     <input type="radio" name="ptype" id="mc" />
22       <label for="mc"><img src="mc.png"
➥/></label>
23     <input type="radio" name="ptype" id="amex" />
24       <label for="amex"><img src="amex.png"
➥/></label>
25     <div id="ccInfo">
26      <input type="text" id="cardNum">Card
➥Number</input>
27       <br><input type="text"
➥id="csc">CSC</input><br>
28      <label>Expires</label><select
➥id="expiresY"></select>
29       <select id="expiresM"></select><br>
30     </div>
31   </form>
32   </body>
33 </html>
```

ch0903.html

Figure 9.3 Automatically focusing on the credit card number when the card type is selected using jQuery code in ch0903.html.

Controlling Form Submission

```
$("form").submit(function(e){
    alert("Sorry. Not yet Implemented.");
    e.preventDefault();
  });
```

Another important aspect of dynamic form flow control is intercepting the `submit` and `reset` event and performing actions based on various conditions. For example, you might want to validate form values before you allow the form to be submitted.

The way that you control the form submission functions to attach a `submit` event handler to the form. Inside the event handler, you have access to information about the event as well as the form data to be submitted. You can perform whatever tasks you need and then either allow the form to be submitted or reset or prevent the default browser action.

The following code illustrates an example of stopping the form submission by calling `.preventDefault()` on the event:

```
$("form").submit(function(e){
    alert("Sorry. Not yet Implemented.");
    e.preventDefault();
  })
```

This page intentionally left blank

10

Building Web Page Content Dynamically

A critical part of using jQuery and JavaScript is the ability to dynamically build content to add to the page. This content may be form elements, images, lists, tables, or whatever type of element fits the need. Adding content to the page dynamically instead of requesting a new page from the server eliminates the server request time and allows you to build only those new pieces instead of everything.

The phrases in this chapter are designed to help you understand the methods of creating and adding content to the web page. They cover appending, prepending, and inserting elements of different types. Also covered is how to remove content from the web page.

Creating HTML Elements Using jQuery

```
var newP = $('<p id="newp" class="nice">new P</p>');
var img = $("<img></img>");
img.attr("src", "images/newImg.jpg");
var opt = $("<option></option>");
opt.html("Option 1");
opt.val(1);
opt.attr("selected",true):
```

An extremely important aspect of jQuery is the ability to create HTML elements from an HTML string. To create an HTML object from a string, simply call `jQuery(htmlString)` or `$(htmlString)`. The string can include attribute settings and content. For example:

```
var newP = $('<p id="newp" class="nice">new
➥Paragraph</p>');
```

Notice that I used single quotes to define the `htmlString`. That allowed me to put in the `id="newp"` and `class="nice"` without having to escape the double quotes. The resulting `newP` object is a jQuery object with a single `<p>` element in its set.

Another big advantage to building HTML is that you can use the `.attr()`, `.html()`, `.prop()`, `.val()`, and `.data()` methods to store additional values and set attributes, properties, and content.

For example, the following code creates an image element and then assigns the `src` attribute:

```
var img = $("<img></img>");
img.attr("src", "images/newImg.jpg");
```

Another example creates a new <option>, assigns a value, and then sets the <option> as selected by setting the value of the <select> to the value of the option:

```
var opt = $("<option></option>");
opt.html("Option 1");
opt.val(1);
select.val(opt.val());
```

Adding Elements to the Other Elements

An important aspect of dynamic web programming is the ability to add content based on user interaction without the need to request a new page from the server. This gives the users a much quicker and more seamless experience. The phrases in this section are designed to illustrate the basics of prepending, appending, and inserting new HTML elements into existing ones.

Prepending to the Beginning of an Element's Content

```
var newDiv = $("<div></div>");
$("body").append(newDiv);
newDiv.append($("<p>Paragraph A</p>"));
$("<p>Paragraph A</p>").appendTo("div");
```

jQuery provides methods to prepend content before the existing content in an element. For example, you can add list items, a table row, or paragraph text before existing list items, table rows, or text in a paragraph.

These methods are `existingObject.prepend(newContent)` and `newContent.prependTo(existingObject)`. The `existingObject` can be an object you already have, or you can use a selector to specify the existing object.

For example, the following code creates a new `<div>` and appends it to `body`. Then it prepends two paragraphs each using a different method:

```
var newDiv = $("<div></div>");
$("body").prepend(newDiv);
newDiv.prepend($("<p>Paragraph A</p>"));
$("<p>Paragraph A</p>").prependTo("div");
```

Appending to the Bottom of an Element's Content

```
var newDiv = $("<div></div>");
$("body").append(newDiv);
newDiv.append($("<p>Paragraph A</p>"));
$("<p>Paragraph B</p>").appendTo("div");
```

jQuery also provides methods to append content after the existing content in an element. For example, you can add list items, table rows, or paragraph text at the end of a list item, table row, or text in a paragraph.

These methods are `existingObject.append(newContent)` and `newContent.appendTo(existingObject)`. The `existingObject` can be an object you already have, or you can use a selector to specify the existing object.

For example, the following code creates a new `<div>` and appends it to `body`. Then it appends two paragraphs each using a different method:

```
var newDiv = $("<div></div>");
$("body").append(newDiv);
newDiv.append($("<p>Paragraph A</p>"));
$("<p>Paragraph A</p>").appendTo("div");
```

Inserting into the Middle of an Element's Content

```
$("ul li:eq(2)).insertBefore($("<li>New
➥ Item</li>"));
$("<p>Title</p>").before("#subtitle");
$("ul li:eq(2)).insertAfter($("<li>New Item</li>"));
$("<p>Sub Title</p>").after("#title");
```

Often, you will want to add new items to the middle of an element's content. For example, you might need to insert an item into the middle of a list or select or add a new paragraph right before an existing paragraph.

jQuery provides methods to insert content immediately before and after existing content. The new content is a sibling of the existing content.

To insert an element before an existing element, use `existingObject.insertBefore(newContent)` or `newContent.before(existingObject)`. The `existingObject` can be an object you already have, or you can use a selector to specify the existing object.

For example, the following code inserts a new list item immediately before the third item already in the list:

```
$("ul li:eq(2)).insertBefore($("<li>New
➥Item</li>"));
```

The following code inserts a new paragraph before one with an `id="subtitle"` using the `.before()` method:

```
$("<p>Title</p>").before("#subtitle");
```

To insert an element after an existing element, use `existingObject.insertAfter(newContent)` or `newContent.after(existingObject)`. The `existingObject` can be an object you already have, or you can use a selector to specify the existing object.

For example, the following code inserts a new list item immediately after the third item already in the list:

```
$("ul li:eq(2)).insertAfter($("<li>New Item</li>"));
```

The following code inserts a new paragraph immediately after one with an `id="title"` using the `.after()` method:

```
$("<p>Sub Title</p>").after("#title");
```

Adding Elements Example

The code that follows appends, inserts, and prepends paragraphs to an existing `<div>` element when the button is clicked so that you can see how the dynamic inserts work. Figure 10.1 illustrates how the page looks before and after adding the elements:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
```

```
07   <script>
08     $(document).ready(function (){
09       $("button").on("click", function(){
10         $("div").prepend("<p>Paragraph 1</p>");
11         $("div p:eq(1)").after("<p>Paragraph
➥3</p>");
12         $("div").append("<p>Paragraph 5</p>");
13       });
14     });
15   </script>
16 </head>
17   <body>
18     <button>Add Paragraphs</button>
19     <div>
20       <p>Paragraph 2</p>
21       <p>Paragraph 4</p>
22     </div>
23   </body>
24 </html>
```

ch1001.html

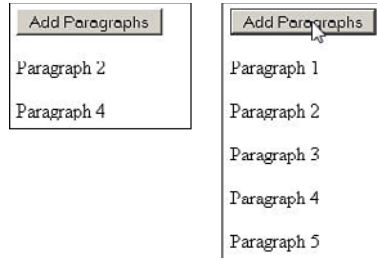


Figure 10.1 Dynamically adding paragraph elements using jQuery code in ch1001.html.

Removing Elements from the Page

```
$("p").remove();
$("p").remove(".menu");
$("#myList").children("li").remove();
```

You remove elements from the web page using the `.remove([selector])` method on the jQuery object. This method modifies the DOM and removes the nodes from the tree.

There are a few different ways to use the `.remove()` method. Calling `.remove()` with no parameters removes all elements in the jQuery object's set. For example, the following statement removes all `<p>` elements:

```
$("p").remove();
```

You can also pass a selector into `.remove()`, which limits the removal to items in the set that match the selector value. For example, the following statement removes all `<p>` elements with `class="menu"`:

```
$("p").remove(".menu");
```

Often, you won't want to remove the element itself, but its children. For example, the following code removes all `<li>` elements in a list with `id="myList"`:

```
$("#myList").children("li").remove();
```

The following code shows an example of using selectors to remove elements from a list. The code first generates a list of ten elements and then provides buttons that call handlers that use `.remove("li:even")` or `.remove("li:odd")` to remove only the even or odd elements from the list, as shown in Figure 10.2:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    $(document).ready(function (){
      for(var i=0; i<10; i++){
        $("ul").append($("<li></li>").html("Item
➥"+ i));
      }
      $("#odd").on("click", function(){
        $("li:odd").remove(); });
      $("#even").on("click", function(){
        $("li:even").remove(); });
    });
  </script>
</head>
  <body>
    <button id="odd">Remove Odd</button>
    <button id="even">Remove Even</button>
    <ul></ul>
  </body>
</html>
```

ch1002.html

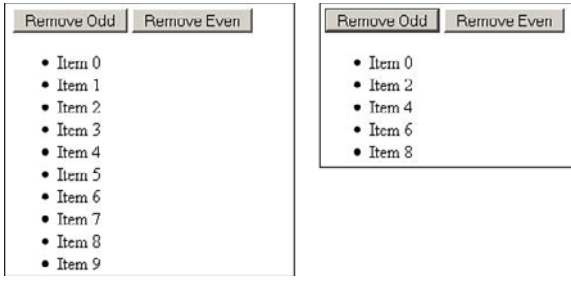


Figure 10.2 Dynamically removing list elements using jQuery code in ch1002.html.

Dynamically Creating a Select Form Element

```
var select = $("<select></select>");
var opt = $("<option></option");
opt.val("1");
opt.html("Option 1");
select.append(opt);
select.val(opt.val());
```

jQuery makes it easy to dynamically generate a select element. This is useful if you are trying to add selects with a large number of options or if the data for the select is coming in a dynamic way, such as user input or an AJAX request.

To dynamically create a select, you first need to create a `<select>` element using the following code:

```
var select = $("<select></select>");
```

Then you need to add options by creating the `<option>` element, setting a value for the option, and

then defining the HTML to appear in the select. The following code creates a new `<option>` and adds it to `<select>`:

```
var opt = $("<option></option");
opt.val("1");
opt.html("Option 1");
select.append(opt);
```

You can also set an option as selected using the following statement to set the value of the select to the value of the option:

```
select.val(opt.val());
```

The following code illustrates an example of dynamically building date selects of month, day, year using `for` loops and a month name array. Figure 10.3 shows the dynamically generated selects:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     var mNames = ["Jan", "Feb", "Mar", "Apr",
➥"May",
09       "Jun", "Jul", "Aug", "Sep", "Oct", "Nov",
➥"Dec" ];
10     $(document).ready(function (){
11       var months = $('<select
➥id="mon"></select>');
12       var days = $('<select id="day"></select>');
13       var years = $('<select
```

```
➥id="year"></select>');
14        for(var i=0; i<mNames.length; i++){
15          var month = $("<option></option");
16          var v = mNames[i];
17          month.val(mNames[i]).html(mNames[i]);
18          months.append(month); }
19        for(var i=1; i<=31; i++){
20          var day = $("<option></option");
21          day.val(i).html(i);
22          days.append(day); }
23        for(var i=2010; i<=2030; i++){
24          var year = $("<option></option");
25          year.val(i).html(i);
26          years.append(year); }
27        $("body").append(months, days, years);
28        $("select").on("change", function(){
29          $("p").html($("#mon").val() +" " +
30            $("#day").val() +", " +
➥$("#year").val());
31        });
32        $("select").attr("size", 3);
33      });
34    </script>
35  </head>
36    <body>
37      <p>Date</p>
38    </body>
39  </html>
```

ch1003.html

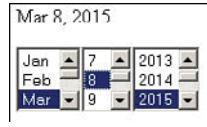


Figure 10.3 Dynamically creating select elements using jQuery and JavaScript code in ch1003.html.

Appending Rows to a Table

```
var newRow = $("<tr></tr>");
var newCell = $("<td>cell</td>");
newRow.append(newCell);
$("tbody").append(newRow):
```

A common task when dynamically manipulating web pages is to add rows to elements in a table. To add rows to a table, you need to create a new `<tr>` element and then add the `<td>` elements. For example:

```
var newRow = $("<tr></tr>");
var newCell = $("<td>cell</td>");
newRow.append(newCell);
$("tbody").append(newRow):
```

The following example illustrates using table building using a series of arrays to populate the data. The `buildTable()` function selects a random item out of the arrays to be placed into the `<td>` elements. Figure 10.4 shows the final table:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     var tArr = ["Mens", "Womens", "Youth",
➥"Childs"];
09     var sArr = ["XL", "M", "S", "XS"];
10     var kArr = ["pants", "shirt", "sweater",
➥"belt"];
11     function randInt(max) {
12       return Math.floor((Math.random()*max)+1); }
```

```
13     function buildTable(){
14       for(var x=1;x<10;x++){
15           var row =$("<tr></tr>");
16           row.append($("<td></td>").html(x));
17           row.append($("<td></td>").html(
18             tArr[randInt(3)]+" "+sArr[randInt(3)]
➥+
19               " " +kArr[randInt(3)]));
20
➥row.append($("<td></td>").html(randInt(20)));
21           row.append($("<td></td>").html(
22             ((Math.random()*80)+5).toFixed(2)));
23           $("tbody").append(row);}
24     }
25     $(document).ready(function (){
26       buildTable()
27     });
28   </script>
29   <style> td {border:.5px dotted black; }
➥</style>
30   </head>
31   <body>
32     <table>
33       <thead><tr>
34         <th class="numeric"><span>ID#</span></th>
35         <th ><span>Product</span></th>
36         <th
➥class="numeric"><span>Quantity</span></th>
37         <th
➥class="numeric"><span>Price</span></th>
38         <td class="spacer"></td>
39       </tr></thead>
40       <tbody></tbody>
41     </table>
42   </body>
43 </html>
```

ch1004.html

ID#	Product	Quantity	Price
1	Childs M belt	20	67.84
2	Womens XS sweater	17	84.95
3	Childs XS belt	15	80.49
4	Youth XS shirt	17	56.82
5	Youth M belt	2	56.31
6	Womens S belt	13	5.91
7	Youth XS sweater	8	10.80
8	Womens S belt	11	62.23
9	Womens S sweater	3	61.91

Figure 10.4 Dynamically adding rows to a table using jQuery and JavaScript code in ch1004.html.

Inserting Items into a List

```
var item = "New Item";
var nextItem = false;
$("ul li").each(function(i, item){
  if ($(item).html() > newItem){
    nextItem = item;
    return false; }});
if (!nextItem){
 $("ul").append($("<li></li>").html(newItem)); }
else {
 $(nextItem).before($("<li></li>").html(newItem));}
```

Another common task when dynamically manipulating web pages is to insert elements into a list. But what if the list needs to be sorted? There are a few different ways to do this.

One way to add the item would be to iterate through the list elements until you find one that is larger than the new item. Then insert the new item before that existing item. If a larger item is not found, just append the item to the list.

The following example illustrates using a text input to allow the user to add items to a grocery list by typing the item and pressing the Enter key. The items are added in alphabetical order in the `addToList()` function. Figure 10.5 illustrates adding items to the list:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    function addToList(newItem){
      var nextItem = false;
      $("ul li").each(function(i, item){
        if ($(item).html() > newItem){
          nextItem = item;
          return false; }
      });
      if (!nextItem){

➥$("ul").append($("<li></li>").html(newItem)); }
      else {

➥$(nextItem).before($("<li></li>").html(newItem));}
    }
    $(document).ready(function (){
      $("input").on("keyup",function(e){
        if (e.keyCode == 13){
          addToList($(this).val());
          $(this).val("");
        }
      });
    });
  </script>
  </head>
```

```
30   <body>
31     <p>Grocery List</p>
32     <input type="text" />
33     <ul><li>eggs</li><li>milk</li></ul>
34   </body>
35 </html>
```

ch1005.html

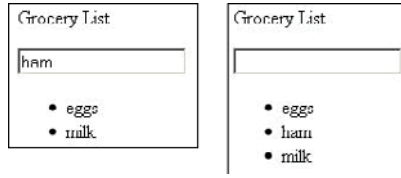


Figure 10.5 Dynamically inserting items alphabetically into a list using jQuery and JavaScript code in ch1005.html.

Creating a Dynamic Image Gallery

```
var img = $("<img></img>");
img.attr("src", "images/newImage.jpg");
$("div").append(img);
```

Images are becoming a mainstay in web pages. They deliver much more content, and they accomplish it more quickly and often more accurately than text. jQuery and JavaScript allow you to add images to a web page dynamically on demand by creating image elements, then setting the `source` attribute of each and then appending them to the correct location.

For example, the following code creates an <img> element, sets the src attribute to define the file location, and then appends the image to a element:

```
var img = $("<img></img>");
img.attr("src", "images/newImage.jpg");
$("div").append(img);
```

As an example, the following code uses an array of image names to dynamically build a small image gallery on the web page. The addImages() function iterates through the image list and then appends the images to the page by creating <img>, , and <p> elements. Figure 10.6 shows the final image gallery:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
  var images = ["arch", "flower", "sunset2",
                "tiger", "falls", "volcano",
                "beach", "img1", "flower2"];
    function addImages(){
      for (i in images){
        var imgbox = $("<span></span>");
        var img = $("<img></img>");
        var label = $("<p></p>").html(images[i]);
        img.attr("src", images[i] + ".jpg");
        imgbox.append(label, img );
```

```
        $("div").append(imgbox); }
    }
    $(document).ready(function (){
      addImages();
    });
  </script>
  <style>
    div, img  { border:5px ridge white;
      box-shadow:10px 10px 5px #888888; }
    div { background-color:black;
      width:660px; height:520px }
    span { display:inline-block; width:200px;
      text-align:center; margin:10px; }
    p {color:white; margin:0px;
      background-color:#3373A5; font-weight:bold;
➥}
    img { height:100px; margin:10px;}
  </style>
  </head>
  <body>
    <div>
    </div>
  </body>
</html>
```

ch1006.html

Figure 10.6 Dynamically generating an image gallery using jQuery and JavaScript code in ch1006.html.

Adding HTML5 Canvas Graphics

```
var c = $("canvas").get(0);
var lineValues = [10,15,80,79];
c.width = c.width;
var xAdj = c.width/lineValues.length;
var ctx = c.getContext("2d");
ctx.fillStyle = "#000000";
ctx.strokeStyle = "#00ff00";
ctx.lineWidth = 3;
var x = 1;
ctx.moveTo(x,(c.height));
for (var idx in lineValues){
  var value = parseInt(lineValues[idx]);
```

```
  ctx.lineTo(x+xAdj, (c.height - value));
  x += xAdj;
}
ctx.stroke();
```

As HTML5 begins to become more and more mainstream, many possibilities open up for graphics. Using dynamic jQuery and JavaScript code with the new `<canvas>` element, you can easily provide some rich, interactive graphics to your web pages.

Using JavaScript, you can access the `<canvas>` element, get the context, set the line colors and the size, and then add lines to your file. To illustrate this, look at the `renderSpark()` function in the following code.

This code accepts a list of values and a `<canvas>` element and then uses the values to draw a series of lines on the canvas. The line values are dynamically generated in the `adjValues()` function and updated on a `setTimeout()` interval. The result is a dynamically updating spark line. You can easily replace the `adjValues()` function with an AJAX request to get server data to populate the canvas graph. Figure 10.7 shows the dynamically rendered graphs on the canvas:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     function randInt(max) {
09       return Math.floor((Math.random()*max)+1); }
10     function adjValues(){
11       $("canvas").each(function(){
```

```
12          var lineValues =
➥$(this).data("valueArr");
13          lineValues.shift();
14          lineValues.push(randInt(100));
15          renderSpark(this, lineValues);
16        });
17        setTimeout(adjValues, 1000);
18      }
19      function renderSpark(c, lineValues){
20        c.width = c.width;
21        var xAdj = c.width/lineValues.length;
22        var ctx = c.getContext("2d");
23        ctx.fillStyle = "#000000";
24        ctx.strokeStyle = "#00ff00";
25        ctx.lineWidth = 3;
26        var x = 1;
27        ctx.moveTo(x,(c.height));
28        for (var idx in lineValues){
29          var value = parseInt(lineValues[idx]);
30          ctx.lineTo(x+xAdj, (c.height - value));
31          x += xAdj;
32        }
33        ctx.stroke();
34      }
35      function getRandomArray(){
36        var arr = new Array();
37        for(var x=0; x<20; x++){
➥arr.push(randInt(100)); }
38        return arr;
39      }
40      $(document).ready(function(){
41        $("canvas").each(function(){
42          $(this).data("valueArr",
➥getRandomArray()); });
43        adjValues();
44      });
45    </script>
```

```
46   <style>
47   canvas{height:50px;width:200px;background-
➥color:black;
48     border:3px solid;vertical-align:bottom;
➥margin:10px;}
49   label, span {display:inline-block; text-
➥align:right;
50     width:160px;border-bottom:2px dotted;font-
➥size:26px;}
51   span{ width:50px; color:blue;
52   </style>
53   </head>
54   <body>
55     <label>Utilization</label><canvas></canvas>
56     <label>Speed</label><canvas></canvas>
57     <label>Uploads</label><canvas></canvas>
58     <label>Downloads</label><canvas></canvas>
59   </body>
60 </html>
```

ch1007.html

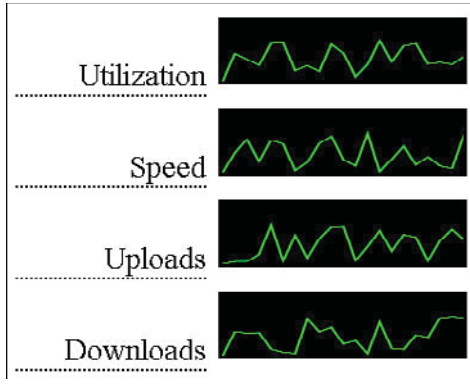


Figure 10.7 Dynamically adding lines to a <canvas> element using JavaScript code in ch1007.html.

This page intentionally left blank

11

Adding jQuery UI Elements

jQuery UI is an additional library built on top of jQuery. The purpose of jQuery UI is to provide a set of extensible interactions, effects, widgets, and themes that make it easier to incorporate professional user interface (UI) elements in your web pages.

The jQuery UI library provides a lot of functionality. The phrases in this chapter are designed to get you started using jQuery UI. Although these concepts are not comprehensive, you can apply them to additional UI elements and interactions.

Adding the jQuery UI Library

```
<script type="text/javascript"
        src="jquery-ui-1.10.3.min.js"></script>
<link rel="stylesheet" type="text/css"
      href="jquery-ui-1.10.3.min.css">
```

To get going with jQuery, download the library, add it to a location where your web pages can see it, and then load the JS and CSS files in the web page. The

process of adding the jQuery UI library is described in the following steps.

1. Download the library from http://jqueryui.com/download/.
2. Unzip the contents of the downloaded files.
3. Copy the `jquery-ui-###.js`, `jquery-ui-###.css`, and `images` folder in the download to a location in your web site where your pages can access them. The `images` folder needs to be in the same location as the `.css` file.
4. Add the following lines to the header of your web pages to load the jQuery UI code. You need to replace the version numbers to match those you downloaded and set the appropriate paths in the `src` and `href` attributes:

```
<script type="text/javascript"
        src="jquery-ui-1.10.3.min.js"></script>
<link rel="stylesheet" type="text/css"
      href="jquery-ui-1.10.3.min.css">
```

By the way

The jQuery UI library is based on the jQuery library, so you need to load the jQuery library first.

Implementing an Autocomplete Input

```
. . . jQuery . . .
$( "#autocomplete" ).autocomplete({
   source: ["Monday", "Tuesday", "Wednesday",
➥"Thursday", "Friday"]
});
. . . HTML . . .
<input id="autocomplete">
```

The autocomplete widget is attached to text input elements. As the user types in the text input, suggestions from a list are displayed. This is especially helpful when you have a finite set of possibilities that can be typed in and you want to make sure the correct spelling is used.

To apply the autocomplete widget, define a source array that contains the strings available to autocomplete in the text input. Then call the `.autoComplete({source:array})` method on the input element. For example:

```
$( "#autocomplete" ).autocomplete({
   source: ["Monday", "Tuesday", "Wednesday",
➥"Thursday", "Friday"]
});
```

The following code illustrates an example of using autocomplete by adding a day array autocomplete widget to an `<input>` element named `#autocomplete`. The autocomplete is illustrated in Figure 11.1. The set of days to autocomplete is added by setting the `source` attribute to an array of day names in lines 16 and 17:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    function dateChanged(dateStr, Object){
       $("span").html("Updated to" + dateStr); }
    $(document).ready(function(){
      $( "#autocomplete" ).autocomplete({
        source: ["Monday", "Tuesday",
➥"Wednesday",
                 "Thursday", "Friday"]
      });
    });
  </script>
  <style>
    input { border:2px ridge blue;
➥border-radius:5px;
      padding:3px; }
  </style>
  </head>
  <body>
    <label for="autocomplete">Day of Week:
➥</label>
    <input id="autocomplete">
  </body>
</html>
```

ch1101.html

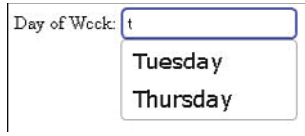


Figure 11.1 Using jQuery UI code to implement an autocomplete text input in ch1101.html.

Implementing Drag and Drop

```
$("#drag1, #drag2, #drag3").draggable(
  {helper:"clone", cursor:"move",
   opacity:.7, zIndex:99});
$("#drop").droppable({accept:"img",
  tolerance:"intersect", hoverClass:"drop-hover"});
$("#drop").on("drop", function(e,ui){
  var item = $("<div></div>");
  item.append($("<img></img>").attr("src",
    ui.draggable.attr("src")));
  item.append($("<span></span>").html(
    ui.draggable.attr("src")));
  $(this).append(item);
});
```

You can define one element to be draggable and then another to be droppable. When draggable elements are dropped on droppable widgets, you can add JavaScript and jQuery code that provides whatever interaction necessary when draggable elements are dropped on droppable widgets.

The draggable widget provides the necessary interactivity for elements when dragging them with the mouse. For example, moving an element to another position in the browser window by dragging it with the mouse.

Table 11.1 describes the more common draggable options. The following code shows an example of

attaching the draggable widget to an element with the `cursor` and `opacity` options:

```
$("#img1").draggable({cursor:"move", opacity:.5});
```

Table 11.1 Common Draggable Widget Options

Option	Description
`axis`	Can be set to `x` or `y` or `false`. `x` drags horizontally only, `y` drags vertically only, and `false` drags freely.
`containment`	Specifies a container to limit dragging within. Possible values are `"parent"`, `"document"`, or `"window"`.
`cursor`	Specifies the cursor to display while dragging.
`helper`	Defines what element is displayed when dragging. Values can be `"original"`, `"clone"`, or a function that returns a DOM object.
`opacity`	Sets the opacity while dragging.
`revert`	Boolean. Specifies if the `"original"` object should return to its original position when dragging stops. String. If set to `"valid"`, revert only occurs if the object has been dropped successfully. `"invalid"` reverts only if the object hasn't been dropped successfully.
`stack`	Is set to `false` or a selector. If a selector is specified, the item is brought to the top z-index of the element the selector specifies.
`zIndex`	z-index value to use while dragging.

The draggable widget also provides the additional events so handlers can be attached to the element when dragging starts, is in progress, and stops. Table 11.2 lists the events that you can access on draggable items. The following code shows an example of adding a `dragstop` event to apply a `bounce` effect when the item is dropped:

```
$("#drag1").draggable({cursor:"move", opacity:.5});
$("#drag1").on("dragstop",
➥function(){$(this).effect("bounce", 1000); });
```

Table 11.2 Draggable Widget Events

Event	Description
`drag(event, ui)`	Triggered while dragging. `event` is the JavaScript event object. `ui` is an object with the following values: ▪ **`helper`**—jQuery object representing the helper for the draggable item. ▪ **`position`**—`{top, left}` object for the current CSS position. ▪ **`offset`**—`{top, left}` object for the current CSS offset.
`dragstart(event, ui)`	Triggered when dragging starts.
`dragstop(event, ui)`	Triggered when dragging stops.

The droppable widget defines an element as a valid drop container usable by draggable items. This allows you to provide interactions between elements using simple mouse controls.

The droppable widget allows you to specify an accept function that can process information about the event, such as mouse coordinates and the draggable item involved. Table 11.3 describes the more common droppable options. The following code shows an example of attaching the droppable widget to an element and specifying the `tolerance` level:

```
$("#div1"). droppable ({tolerance:"touch"});
```

Table 11.3 Common Droppable Widget Options

Option	Description
`accept`	Specifies a selector used to filter the elements that the droppable item will accept.
`activeClass`	Specifies a class that will be applied to the droppable item while a valid draggable item is being dragged.
`greedy`	Boolean. The default is `false`, meaning that all valid parent droppable items receive the draggable item as well. When `true`, only the first droppable item receives the draggable item.
`hoverClass`	Specifies a class to be applied to the droppable item while a valid draggable item is hovering over it.
`tolerance`	Specifies the method used to determine if a draggable item is valid. Acceptable values are ▪ **`fit`**—Draggable entirely overlaps droppable. ▪ **`intersect`**—Draggable overlaps droppable at least 50% in both directions.

- **pointer**—Mouse is over droppable.
- **touch**—Draggable overlaps the droppable element anywhere.

The droppable widget also provides the additional events so handlers can be attached to the element when dragging and dropping. Table 11.4 lists the events that you can access on droppable items. The following code shows an example of adding a `dropactivate` event to apply a `shake` effect when a drag start activates a droppable item:

```
$("#drop1").droppable({tolerance:"pointer"});
$("# drop1").on("dropactivate",
➥function(){$(this).effect("shake", 1000); });
```

Table 11.4 Droppable Widget Events

Event	Description
`dropactivate(event, ui)`	Triggered when a valid draggable item begins dragging. `event` is the JavaScript event object. `ui` is an object with the following values: - **draggable**—jQuery object representing the actual draggable item. - **helper**—jQuery object representing the helper for the draggable item.

	■ **position**—{top, left} object for the current draggable CSS position. ■ **offset**—{top, left} object for the current draggable CSS offset.
drop(event, ui)	Triggered when a draggable item is dropped on droppable.
dropout(event, ui)	Triggered when draggable leaves droppable based on tolerance.
dropover(event, ui)	Triggered when draggable enters droppable based on tolerance.

Now I'll try to put all this together in a practical way. The following code makes three image elements draggable and then a element droppable. I use the "clone" option in line 14 so that I can move a copy of the image and leave the original in place. The drop event handler uses the image information to create a new element that contains the image and filename, as shown in Figure 11.2:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script type="text/javascript"
08     src="../js/jquery-ui-1.10.3.min.js"></script>
09   <link rel="stylesheet" type="text/css"
10     href="../js/css/jquery-ui-1.10.3.min.css">
```

```
11   <script>
12     $(document).ready(function(){
13       $("#drag1, #drag2, #drag3").draggable(
14         {helper:"clone", cursor:"move",
15          opacity:.7, zIndex:99});
16       $("#drop").droppable({accept:"img",
17         tolerance:"intersect", hoverClass:"drop-
➥hover"});
18       $("#drop").on("drop", function(e,ui){
19         var item = $("<div></div>");
20         item.append($("<img></img>").attr("src",
21           ui.draggable.attr("src")));
22         item.append($("<span></span>").html(
23           ui.draggable.attr("src")));
24         $(this).append(item);
25       });
26     });
27   </script>
28   <style>
29     div {display:inline-block; vertical-
➥align:top;
30       margin:10px;}
31     img {width:100px; margin:0px; }
32     #images { width:100px; height:300px; }
33     #drop { width:300px; min-height:150px;
➥padding:3px;
34       border:3px ridge white; box-shadow:
35       5px 5px 5px #888888; }
36     #drop div{ height:80px; width:280px;
➥padding:4px;
37       border:2px dotted; margin-top:5px; }
38     #drop div img {height:80px; margin-
➥right:10px; }
39     #drop div span { display:inline-block;
40       vertical-align:top; font:16px/70px arial; }
41     .drop-hover { background-color:#BBDDFF; }
42   </style>
```

```
43   </head>
44   <body>
45     <div id="images">
46       <img id="drag1" src="lake.jpg" />
47       <img id="drag2" src="tiger.jpg" />
48       <img id="drag3" src="volcano.jpg" />
49     </div>
50     <div id="drop"></div>
51   </body>
52 </html>
```

ch1102.html

Figure 11.2 Applying draggable and droppable to elements using jQuery UI code in ch1102.html.

Adding Datepicker Element

```
$( "#month" ).datepicker({
    onSelect:dateChanged,
    showOn: "button",
    buttonImage: "calendar.png",
    buttonImageOnly: true,
    numberOfMonths:2,
    showButtonPanel:true,
    dateFormat: "yy-mm-dd"
  });
```

The datepicker widget provided with jQuery allows you to implement a calendar interface so users can select a specific day using a simple click of the mouse.

This can prevent a lot of problems when users input dates incorrectly because they are typing them by hand.

The datepicker widget is attached to a `text`, `date`, or `datetime` `<input>` element by calling the `.datepicker()` method. When the user clicks on the `<input>`, the calendar is displayed. You can also add an icon image to launch the datepicker using the `buttonImage` attribute.

The following code creates a date with an image icon, as seen in Figure 11.3. To illustrate some of the available options, I've added settings for the following:

- **`onSelect`**—Specifies a function that is called each time a new date is selected.
- **`showOn`**—This is set to "button" so that the datepicker is launched when the button icon is clicked.
- **`buttonImage`**—Specifies the location of the image file to use.
- **`buttonImageOnly`**—When true, the datepicker is only launched when the button icon is clicked and not the `<input>`.
- **`numberOfMonths`**—Specifies the number of months to display.
- **`showButtonPanel`**—When `true`, the Today and Done buttons are displayed on the bottom of the datepicker.
- **`dateFormat`**—String that describes the format to be placed in the `<input>` field.

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    function dateChanged(dateStr, Object){
      $("span").html("Updated to" + dateStr); }
    $(document).ready(function(){
      $( "#month" ).datepicker({
          onSelect:dateChanged,
          showOn: "button",
          buttonImage: "calendar.png",
          buttonImageOnly: true,
          numberOfMonths:2,
          showButtonPanel:true,
          dateFormat: "yy-mm-dd"
        });
    });
  </script>
  <style>
    input { border:2px ridge blue; border-radius:
➥5px; }
  </style>
  <body>
    <label>Date: </label>
    <input type="text" id="month"></input>
    <span></span>
  </body>
</html>
```

ch1103.html

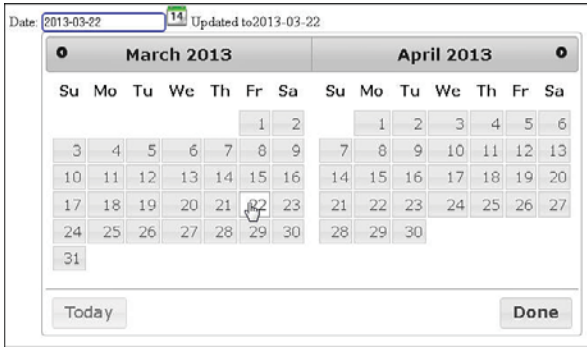


Figure 11.3 Implementing a datepicker widget in jQuery UI code in ch1103.html.

Using Sliders to Manipulate Elements

```
$( "#red, #green, #blue" ).slider({
  orientation: "horizontal",
  range: "min",
  max: 255,
  value: 127,
  slide: refreshSwatch,
  change: refreshSwatch
});
$("#red").slider("value", 128);
$("#green").slider("value", 128);
$("#blue").slider("value", 128);
```

The slider widget allows you to create slider controls that adjust a value by dragging the mouse. The slider has two components: the slide and the handle. The slide is styled by the `.ui-slider-range` class, and the handle is styled by the `.ui-slider-handle` class.

Did you know?

You can define both a min and a max handle that allows you to use a single slider control to define a range instead of a single value.

The following code provides an example of implementing a set of sliders to adjust the background color of another element. The slider is applied to the `<div>` elements in lines 23–33, and the following options are set:

- **`orientation`**—Can be set to `"horizontal"` or `"vertical"`.
- **`range`**—Can be set to `true`, `"min"`, or `"max"`. Used to define the range. `"min"` goes from the slider min to a handler, and `"max"` goes from the slider max to a handle.
- **`max`**—Specifies the maximum value.
- **`value`**—Specifies the current value.
- **`slide`**—Event handler to call when the slide moves.
- **`change`**—Event handler to call when the slide value changes.

Also, pay attention to the class settings in lines 41–46 of the CSS, which alter the appearance of the slider and handler. Figure 11.4 shows the sliders:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    function cValue(selector){
      var v =
➥$(selector).slider("value").toString( 16 );
      if (v.length ===1) { v = "0" + v;}
      return v;
    }
    function refreshSwatch() {
      $("#mix").css("background-color", "#" +

➥cValue("#red")+cValue("#green")+cValue("#blue"));
      $("#mix").html($("#mix").css("background-
➥color"));
    }
    $(document).ready(function(){
       $( "#red, #green, #blue" ).slider({
        orientation: "horizontal",
        range: "min",
        max: 255,
        value: 127,
        slide: refreshSwatch,
        change: refreshSwatch
      });
      $("#red").slider("value", 128);
```

```
32        $("#green").slider("value", 128);
33        $("#blue").slider("value", 128);
34      });
35    </script>
36    <style>
37      #mix { width:160px; height:100px; text-
➥align:center;
38        font:18px/100px arial; }
39      #red, #green, #blue { float: left; clear:
➥left;
40        width:150px; margin:15px;    }
41      #red .ui-slider-range { background:red; }
42      #red .ui-slider-handle { border-color:red; }
43      #green .ui-slider-range { background:green; }
44      #green .ui-slider-handle { border-
➥color:green; }
45      #blue .ui-slider-range { background:blue; }
46      #blue .ui-slider-handle { border-color:blue;
➥}
47    </style>
48    <body>
49      <div id="mix"></div>
50      <div id="red"></div>
51      <div id="green"></div>
52      <div id="blue"></div>
53    </body>
54 </html>
```

ch1104.html

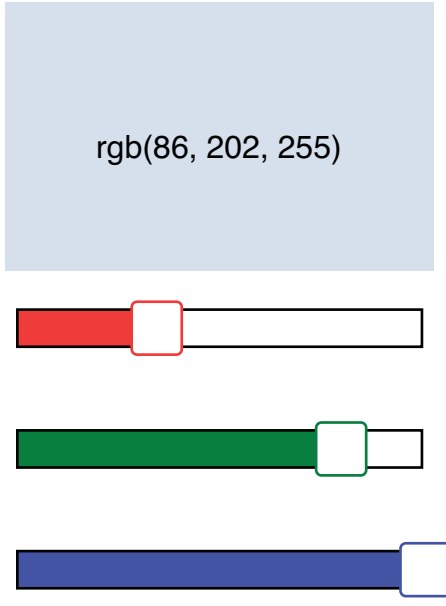


Figure 11.4 Adding sliders in jQuery UI code in ch1104.html.

Creating a Menu

```
. . . jQuery . . .
$( "#menu" ).menu();
$( "#menu" ).on("menuselect", function(e, ui){
  $("p").html("Selected " +
    ui.item.children("a:first").html());
});
. . . HTML . . .
<ul id="menu">
  <li><a href="#">Open</a></li>
  <li><a href="#">Recent</a><ul>
. . .
  <li class="ui-state-disabled"><a href="#">
```

One of the most often used jQuery UI widgets is the menu widget. The menu widget allows you to turn an element tree into an expanding menu. Typically, menus are created by using cascading sets of `<ul>/<li>` elements with an `<a>` element that defines the link behavior and menu text.

Did you know?

You can customize the element tags that are used to build the element using the `menus` option; for example: `menus:"div.menuItem"`.

The following code creates a jQuery UI menu from a set of lists. Notice in the HTML that some of the `<li>` fields include a `<span>` that has `class="ui-icon ui-icon-{type}"`. These items will include the jQuery UI icon specified along with the menu text.

The selected item is displayed in the `<p>` element to show how the selection handler works using the `menuselect` event handler defined in line 14. The width of the menu is defined in the CSS code on line 21 by setting the `width` value in the `.ui-menu` class. The menu action is illustrated in Figure 11.5:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script type="text/javascript"
08     src="../js/jquery-ui-1.10.3.min.js"></script>
09   <link rel="stylesheet" type="text/css"
10     href="../js/css/jquery-ui-1.10.3.min.css">
```

```
<script>
  $(document).ready(function(){
    $( "#menu" ).menu();
    $( "#menu" ).on("menuselect", function(e,
➥ui){
      $("p").html("Selected " +
        ui.item.children("a:first").html());
    });
  });
</script>
<style>
 .ui-menu { width: 200px; }
 p { border:3px ridge red; color:red;
     display:inline-block; height:80px;
➥width:100px; }
</style>
<body>
  <ul id="menu">
    <li><a href="#">Open</a></li>
    <li><a href="#">Recent</a><ul>
      <li><a href="#">Some File</a></li>
      <li><a href="#">Another File</a></li>
      </ul></li>
    <li><a href="#">Save</a></li>
    <li class="ui-state-disabled"><a href="#">
      <span class="ui-icon ui-icon-print">
      </span>Print...</a></li>
    <li><a href="#">Playback</a><ul>
      <li><a href="#">
       <span class="ui-icon ui-icon-seek-
➥start"></span>
        Prev</a></li>
      <li><a href="#">
        <span class="ui-icon ui-icon-
➥stop"></span>
        Stop</a></li>
      <li><a href="#">
```

```
44           <span class="ui-icon ui-icon-
➥play"></span>
45           Play</a></li>
46         <li><a href="#">
47           <span class="ui-icon ui-icon-seek-
➥end"></span>
48           Next</a></li>
49         </ul></li>
50     </ul>
51     <p></p>
52   </body>
53 </html>
```

ch1105.html

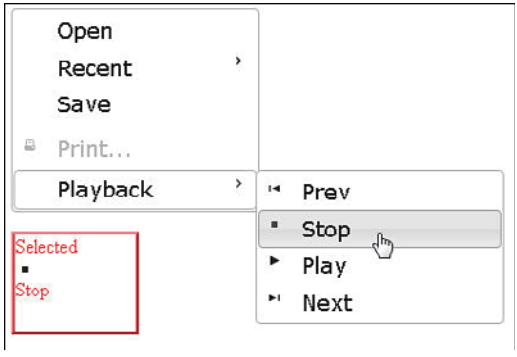


Figure 11.5 Implementing a jQuery UI menu using the code in ch1105.html.

Adding Tooltips

```
$(document).tooltip({
  items: "img, input",
  position: {my:"left+15 top", at:"left bottom",
             collision:"flipfit" },
  content: function() {
    var obj = $(this);
    if (obj.is("input")) {
      return obj.attr( "title" ); }
    if (obj.is("img")) {
      var img = $("<img></img>").addClass(
        "mini").attr("src", obj.attr("src"));
      var span = $("<span></span>").html(
        obj.attr( "alt" ));
      return $("<div></div>").append(img, span); }
  }});
```

The tooltips widget allows you to easily add tooltips to form input, images, and just about any other page element. To implement the tooltips, apply `.tooltip(options)` to the document or other container. Inside the `options`, specify the items that should include tooltips and then the tooltip content handler.

As the mouse hovers over an item supported by the tooltip, the tooltip message is displayed. The following code provides an example of implementing tooltips on `<input>` and `<img>` elements, as shown in Figure 11.6.

- **`items`**—Specifies the selector used to determine if the page element supports tooltips.
- **`content`**—Tooltip handler function called when a supported element is hovered over. The function should return the content to be displayed. Notice that for the image, a mini version is displayed in the tooltip.
- **`position`**—Specifies the position to place the tooltip; for example:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script type="text/javascript"
08     src="../js/jquery-ui-1.10.3.min.js"></script>
09   <link rel="stylesheet" type="text/css"
10     href="../js/css/jquery-ui-1.10.3.min.css">
11   <script>
12     $(document).ready(function(){
13       $(document).tooltip({
14         items: "img, input",
15         position: {my:"left+15 top", at:"left
➥bottom",
16                    collision:"flipfit" },
17         content: function() {
18           var obj = $(this);
19           if (obj.is("input")) {
20             return obj.attr( "title" ); }
21           if (obj.is("img")) {
22             var img = $("<img></img>").addClass(
23               "mini").attr("src",
➥obj.attr("src"));
24             var span = $("<span></span>").html(
25               obj.attr( "alt" ));
26             return $("<div></div>").append(img,
➥span); }
27         }});
28     });
29   </script>
30   <style>
31     input { border:2px ridge blue; border-radius:
➥5px;
32       padding:3px; }
33     img { height:200px; margin:15px; }
```

```
34      .mini {height:30px;}
35    </style>
36    <body>
37      <label for="size">Who are you? </label>
38      <input id="size"
39        title="Nosce te ipsum (Know Thyself)"
➥/><br>
40      <img src="someday.jpg"
41        alt="I'm going there someday"/>
42    </body>
43 </html>
```

ch1106.html

Figure 11.6 Adding tooltips to elements using jQuery UI code in ch1106.html.

This page intentionally left blank

12

Animation and Other Special Effects

One of the most important features of jQuery is the ability to add animations to changes you are making to elements. This gives the user the feel of a slick, well-developed application rather than a clunky, traditional web page.

This is especially true if you are moving, resizing, or dynamically adding elements to the web page. It is frustrating as a user to all of the sudden have a bunch of new things appear or disappear. Using transitions gives the user a chance to see where elements are expanding from, shrinking to as well as the direction of movement so he can adjust his mind-set to accept the changes.

Understanding jQuery Animation

jQuery animation is the process of modifying the property of an HTML element from one value to another in an incremental fashion that's visible to the user. This section describes that process and how to implement animations on CSS attributes.

Animating CSS Settings

```
$("img").animate({height:100, width:100});
$("p").animate({margin-left:30});
```

Most animation in jQuery is done via the `.animate()` method. The `.animate()` jQuery method allows you to implement animations on one or more CSS properties. Keep in mind that the `.animate()` method acts on all elements in the jQuery object set at the same time. Often, you only want to act on a single element, so you need to filter the set down to one.

The `.animate()` method accepts a CSS property object mapping as the first argument. You can specify more than one property in the object to animate multiple properties at the same time. For example, the following code animates the `height` and `width` properties for `<img>` elements:

```
$("img").animate({height:100, width:100});
```

By the way

The `.animate()` method can only animate properties that have a numerical value. For example, you cannot animate a border style, but you *can* animate a border size.

There are a couple of different ways to call the `.animate()` method. The following shows the syntax of both:

```
.animate(properties [, duration] [, easing] [,
➥complete])
.animate(properties, options)
```

The first method allows you to specify the `duration`, `easing`, and `complete` functions as optional arguments. The second method allows you to specify the options as a single option map object. For example, the following calls `.animate()` with a `duration` and `easing` object map:

```
$("img").animate({height:100, width:100},
  {duration:1000, easing:"linear"});
```

Did you know?

You cannot animate color changes using the color names, but you can animate color changes using hex values, such as `#ff0000`.

Table 12.1 describes the different options available for the `.animate()` method. These options are also available on some of the other animation methods that will be discussed later in this chapter.

Table 12.1 Animation Options

Option	Description
`complete`	Defines a function to be called when the animation has completed.
`duration`	A string or number specifying how long the animation will run. The optional string values are `"slow"` or `"fast"`. A number specifies the number of milliseconds the animation will run. If no duration is specified, the default is 400ms.
`easing`	A string indicating which easing function to use. jQuery provides the `"swing"` (default) and `"linear"` easing function. jQuery UI add several more. Figure 12.1, from jqueryui.com, shows the easing graphs. Think of the horizontal axis of the graphs as duration time, where left is 0s and the right is complete. Think of the vertical axis of the graph as the percentage of change to the setting(s) where the bottom is 0% and the top is 100%.
`queue`	Can be `true`, meaning the animation will be queued up behind any others for the object; `false`, meaning that the animation will start immediately; or a string specifying the name of a specific queue.
`step`	Specifies a function to be executed each step in the animation until the animation completes.

Option	Description
`specialEasing`	You can also map the easing directly in the properties map, allowing you to do different easing for different elements. For example: `$("img").animate({height:[100, ➥"swing"], width:[100,"linear"]}, ➥1000);`

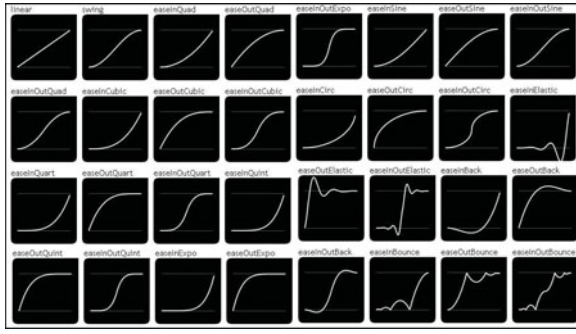


Figure 12.1 jQuery and jQuery UI easing functions.

Understanding Animation Queues

Animations happen asynchronously with code executing, meaning that the code continues to execute while the animation is happening. So what happens if you specify another animation for an object before the first one completes? The answer is that jQuery queues up the animations and then executes them in order one after another until all are competed. That is, unless you specify `queue:false` in the animation options.

You need to understand the animation queue because if you allow user interactions to queue up too many animations by moving the mouse or clicking, the animations will be sluggish and behind the users' actions.

Watch out!

You need to pay attention to where you trigger your animations from. Remember that events will bubble up. If you execute the same animation from all levels during the bubble up phase, you can have some seriously undesired results. To prevent this, you can use the `stopPropagation()` and `stopImmediatePropagation()` methods.

Stopping Animation

```
$("*").stop();
$("#img1").stop(true);
$("img").stop(true, true);
```

jQuery allows you to stop the animations currently executing on elements contained in the jQuery object set. The `.stop([clearQueue] [, jumpToEnd])` method allows you stop animations in a few different ways.

Calling `.stop()` with no parameters simply pauses the animations that are in the queue. The next animation that starts begin executing the animations in the queue again. For example, the following code pauses all animations:

```
$("*").stop();
```

Calling `.stop(true)`, with the `cleareQueue` option set to `true` stops animations at the current point and removes all animations from the queue. For example, the

following stops all animations on images and removes the animations from the queue:

```
$("img").stop(true);
```

Calling `.stop(true, true)`, with the `jumpToEnd` option set to `true`, causes the currently executing animation to jump to the end value immediately, clear the queue, and then stop all animations. For example, the following stops all animations on images but finishes the adjustment made by the current animation and then removes the animations from the queue:

```
$("img").stop(true, true);
```

The `.stop()` method returns the jQuery object, so you can chain additional methods onto the end. For example, the following code stops all animations on images and then starts a new one to set the opacity to .5:

```
$("img").stop(true, true).animate({opacity:.5},
➥1000);
```

Delaying Animation

```
$("img").animate({width:500},
  1000).delay(2000).animate({
    opacity:1} 1000);
```

A great option when implementing animations is adding a delay before the animation is added to the queue. You can use this to provide animations in a more advanced way because you delay the execution of the animation queue, giving the user a better visual experience.

The `.delay(duration, [, queueName])` method allows you to specify the delay duration in milliseconds as well as an optional `queueName` that specifies which queue to apply the delay to. For example, the following code adds a size animation to images. Once the size is complete, there is a delay of 2 seconds, and the opacity animates up to 1:

```
$("img").animate({width:500}, 1000).delay(2000).
➥animate({opacity:1} 1000);
```

By the way

The `.delay()` method is great for delaying between queued jQuery effects; however, it is not a replacement for the JavaScript `setTimeout()` function, which may be more appropriate for certain use cases—especially when you need to be able to cancel the delay.

Animating Visibility

jQuery also provides animation capability in fade methods attached to the jQuery objects. In the end, the fade methods are equivalent to using `.animate()` on the opacity property.

The following sections describe applying animation to the various fading methods.

Fading Elements In

```
$("img").fadeIn(1000, "swing");
```

The `.fadeIn( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and

callback options that allow you to animate fading the opacity of an element from its current value to 1.

For example, the following code applies an animation of 1 second with swing easing to all image elements:

```
$("img").fadeIn(1000, "swing");
```

Fading Elements Out

```
$("img").fadeOut(1000, "swing", function() {
  $(this).fadeIn(1000);});
```

The .fadeIn([duration] [, easing] [, callback]) method provides the optional duration, easing, and callback options, allowing you to animate fading the opacity of an element from its current value to 0.

For example, the following code applies an animation of 1 second with swing easing to all image elements. Then, when completed, it fades them back in again:

```
$("img").fadeOut(1000, "swing", function() {
$(this).fadeIn(1000);});
```

Toggling Element Visibility On and Off

```
$("img").fadeToggle(3000, "swing");
```

The .fadeToggle([duration] [, easing] [, callback]) method provides the optional duration, easing, and callback options, allowing you to animate fading the opacity of an element from its current value to 0, depending on its current value.

For example, the following code applies an animation of 3 seconds with `swing` easing to all image elements. Images that are currently visible are faded out, and images that are currently transparent are faded in:

```
$("img").fadeToggle(3000, "swing");
```

Fading to a Specific Level of Opacity

```
$("img").fadeTo(2000, .5);
$("img").fadeTo(2000, 1);
$("img").fadeTo(2000, .1);
```

The `.fadeTo( duration, opacity [, easing] [, callback])` method provides the `duration` and `opacity` options that specify a specific opacity to end at and how long to animate the transition. It also provides optional `easing` and `callback` arguments.

For example, the following code applies an animation of 2 seconds for all images to transition from the current opacity to .5:

```
$("img").fadeTo(2000, .5);
```

Adding a Transition When Changing Images

```
. . . jQuery . . .
$("img").on("click", function(){
  $("#top").fadeToggle(2000);
});
. . . HTML . . .
<img id="bottom" src="lake.jpg" />
<img id="top" src="volcano.jpg" />
```

You cannot directly animate the transition from one image to another so that one image is fading in while the other is fading out. The way to overcome that challenge is to have two images—one on top of the other—and then animate fading the top one out.

The following code does just that. There are two images with fixed positioning. When you click on the top image, it toggles the image opacity in and out, as shown in Figure 12.2.

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    $(document).ready(function(){
      $("img").on("click", function(){
        $("#top").fadeToggle(2000);
      });
    });
  </script>
  <style>
    img { position:fixed; height:300px;
       border:3px ridge white;
       box-shadow:5px 5px 5px #888888;
       margin:10px; background-color:black;}
  </style>
  </head>
  <body>
    <div>
```

```
27        <img id="bottom" src="lake.jpg" />
28        <img id="top" src="volcano.jpg" />
29      </div>
30    </body>
31 </html>
```

ch1201.html

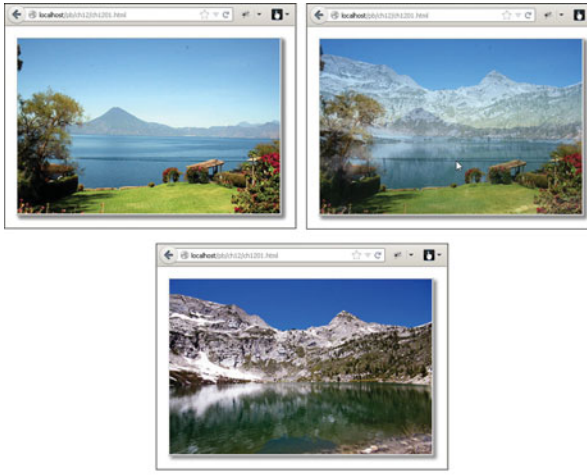


Figure 12.2 Animating transitions between images using jQuery code in ch1201.html.

Making an Element Slide Back to Disappear

```
$("span").mouseover(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:100}, 1000); });
$("span").mouseout(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:.1}, 1000); });
```

```
$("#container").mouseenter(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000); });
$("#container").mouseleave(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000); });
```

The `.fadeTo( duration, opacity [, easing] [, callback])` , `.fadeTo( duration, opacity [, easing] [, callback])`, and `.fadeTo( duration, opacity [, easing] [, callback])` methods provide the `duration`, `easing`, and `callback` arguments that allow you to animate sliding effects in the vertical direction.

For example, the following code applies an animation of 1 second to slide an element down and then a delay of 3 seconds to slide the element back up:

```
$("#menu").slideDown(1000).delay(3000).slideUp(1000);
```

You can animate the `.slideToggle()` method in a similar fashion. For example, the following code animates visibility of a `<div>` element using a slide animation:

```
$("div").slideToggle(1000);
```

I like to use the width and height properties to create a sliding element. You can create a vertical slide animation by animating the height and a horizontal slide animation by animating the width.

There are a couple of tricks. You need to provide both a width and a height value for the element if you want to have the full slide effect and not just a resize effect. Also, if you want the element to maintain space on the page, you cannot animate the value all the way down to 0. However, you can animate down to .1 and the other dimension will retain its space.

The following example shows a slide down animation by changing the height to 100 and then back up by changing the height to .1:

```
$("img").animate({height:100}, 1000);
$("img").animate({height:.1}, 1000);
```

The following code creates a simple web page with a title bar that has a sliding effect revealing an image menu, as shown in Figure 12.3. As you hover over each item in the menu, an image slides down and then slides back up as you leave the menu:

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    $(document).ready(function(){
      $("div div").hide();
      $("span").mouseover(function(){
        var i = $(this).index("span");
        $("img").eq(i).animate({height:100},
➥1000); });
      $("span").mouseout(function(){
        var i = $(this).index("span");
        $("img").eq(i).animate({height:.1},
➥1000); });
      $("#container").mouseenter(function(e){
          e.stopPropagation();
```

```
22
➥$("#images").stop(true).slideToggle(1000); });
23       $("#container").mouseleave(function(e){
24           e.stopPropagation();
25
➥$("#images").stop(true).slideToggle(1000); });
26     });
27   </script>
28   <style>
29     img{ display:inline-block; width:100px;
➥height:.1px;
30       margin:0px; padding:0px; float:left;}
31     p, span { display:inline-block; width:400px;
32       background-color:black; color:white;
33       margin:0px; padding:0px; text-align:center;
34     }
35     span {width:100px; margin:-1px; float:left;
36       border:1px solid; background-color:blue; }
37     #container { width:410px; }
38   </style>
39   </head>
40   <body>
41     <div id="container">
42       <p>Images</p>
43       <div id="images">
44         <span>Image 1</span><span>Image 2</span>
45         <span>Image 3</span><span>Image
➥4</span><br>
46         <img src="img1.jpg" /><img src="img3.jpg"
➥/>
47         <img src="img5.jpg" /><img src="img7.jpg"
➥/>
48       </div>
49     </div>
50   </body>
51 </html>
```

ch1202.html

Figure 12.3 Animating expanding and collapsing menus using sliding techniques in jQuery code in ch1202.html.

Animating Show and Hide

You have already seen the `.show()` and `.hide()` methods in action in Chapter 9, "Dynamically Working with Form Elements." It is common practice to animate this functionality, so jQuery has nicely provided animation options for these methods to make your life easier.

Animating `hide()`

```
$("#box").hide(1000, "linear", function() {
  $("#label").html("Hidden!") });
```

The `.hide( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and `callback` options allowing you to animate the hide effect, making less of a jump when the element disappears.

For example, the following code applies an animation of 1 second with `linear` easing and executes a simple callback function when hiding an element:

```
$("#box").hide(1000, "linear", function() {
   $("#label").html("Hidden!") });
```

Animating show()

```
$("#box").show(1000, "linear", function() {
   $("#label").html("Shown!") });
```

The .show([duration] [, easing] [, callback]) method provides the optional duration, easing, and callback options, allowing you to animate the show effect and make an easier transition as an element appears.

For example, the following code applies an animation of 1 second with linear easing and executes a simple callback function when showing an element:

```
$("#box").show(1000, "linear", function() {
   $("#label").html("Shown!") });
```

Animating toggle()

```
if ($("#handle").html() == '+'){
  $("#photo").show(1000, function(){
    $("#footer").toggle();});
  $("#handle").html('-');
} else {
  $("#footer").toggle();
  $("#photo").hide(1000);
  $("#handle").html('+'); }
```

The .toggle([duration] [, easing] [, callback]) method provides the optional duration, easing, and callback options, allowing you to animate the toggle between the show and hide effect.

For example, the following code applies an animation of 1 second with linear easing and executes a simple callback function when toggling an element between hidden and shown:

```
$("#switch").show(1000, "linear", function() {
  $("#label").html("Switch Toggled!") });
```

The following code implements the `.hide()`, `.show()`, and `.toggle()` and creates a simple web element that provides a title bar with a collapse and expand button on the left so you can expand and collapse an image. Figure 12.4 illustrates the animation.

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    function toggleImage(){
      if ($("#handle").html() == '+'){
        $("#photo").show(1000, function(){
          $("#footer").toggle();});
        $("#handle").html('-');
      } else {
        $("#footer").toggle();
        $("#photo").hide(1000);
        $("#handle").html('+');
      }
    }
    $(document).ready(function(){
      $("#handle").click(toggleImage);
    });
  </script>
  <style>
    div{ width:200px; text-align:center; }
```

```
29      #bar, #handle, #footer{ font-weight:bold;
30        background-color:blue; color:white; }
31      #handle{ display:inline-block;
➥cursor:pointer;
32        background-color:black; width:15px;
➥float:left; }
33      #footer{ font-size:10px; background-
➥color:black; }
34    </style>
35    </head>
36    <body>
37      <div>
38        <div id="bar"><span id="handle">-</span>
39          <span>Image</span></div>
40        <img id="photo" src="img3.jpg"
➥width="200px"/>
41        <div id="footer">Image Ready</div>
42      </div>
43    </body>
44 </html>
```

ch1203.html

Figure 12.4 Animating showing and hiding image elements in jQuery code in ch1203.html.

Animating Resizing an Image

```
$("img").mouseover(function(){
    $(this).animate(
      {width:"200px", opacity:1}, 1000); });
$("img").mouseout(function(){
    $(this).animate(
      {width:"100px", opacity:.3}, 1000); });
```

Similar to the way you use the height and width to create a sliding effect, you can use them to create a resize animation. The difference between a slide and a resize is that the aspect ratio of the image is maintained on a resize, so the element appears to be growing or shrinking rather than being unfolded or untucked.

To create a resize animation, you need to specify both height and width in the `.animate()` statement, or one of them has to be auto in the CSS settings and you can only animate the one that has a value.

The following code shows a resize animation of an image up to 500 pixels over 1 second and then slowly over 5 seconds back down to 400 pixels:

```
$("img").animate({height:500, width:500}, 1000).
➥animate(
  {height:500, width:500}, 5000);
```

The following code creates a basic image gallery view that resizes images and applies opacity changes in the same animation as the mouse enters and leaves the image (see Figure 12.5).

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script type="text/javascript"
    src="../js/jquery-ui-1.10.3.min.js"></script>
  <link rel="stylesheet" type="text/css"
    href="../js/css/jquery-ui-1.10.3.min.css">
  <script>
    $(document).ready(function(){
      $("img").mouseover(function(){
          $(this).animate(
            {width:"200px", opacity:1}, 1000);
➥});
      $("img").mouseout(function(){
          $(this).animate(
            {width:"100px", opacity:.3}, 1000);
➥});
    });
  </script>
  <style>
    div{ padding:0px; }
    img{ opacity:.3; width:100px; float:left; }
  </style>
  </head>
  <body>
    <div id="photos">
      <img src="img1.jpg"/><img src="img2.jpg"/>
      <img src="img3.jpg"/><img src="img4.jpg"/>
      <img src="img5.jpg"/>
    </div>
  </body>
</html>
```

ch1204.html

Figure 12.5 Animating resizing an image using jQuery code in ch1204.html.

Animating Moving an Element

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10,
    top:position.left+100}, 1000);
```

Another dynamic that is good to animate is repositioning of elements—specifically, moving an element from one location to another. Users hate it when they do something and page elements suddenly appear in a different location. Animating the move allows users to see where things go and make the necessary adjustments in their thinking.

To apply move animations, you will be animating changes to the `top` and `left` CSS properties. To animate movement vertically, you use `top`, and to animate horizontally, you use `left`. For example, the following statements animate moving an element to the right 10 pixels and then down 10 pixels:

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10}, 1000);
$("#element").animate({top:position.left+10}, 1000);
```

You can also animate in a diagonal direction by animating both `top` and `left` at the same time. For example, the following animates movement down 10 pixels and to the right 100 pixels in the same animation:

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10,
top:position.left+100}, 1000);
```

The following code shows an example of employing a move animation for a pointless purpose, although it does illustrate the use of this phrase. When the button, shown in Figure 12.6, is hovered over, a move animation moves the element to a new location on the screen, making it impossible to click on:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script type="text/javascript"
08     src="../js/jquery-ui-1.10.3.min.js"></script>
09   <link rel="stylesheet" type="text/css"
10     href="../js/css/jquery-ui-1.10.3.min.css">
11   <script>
12   function randInt(max, min) {
13       return Math.floor((Math.random()*max)+min);
➥}
14     $(document).ready(function(){
15       $("span").mouseover(function(){
```

```
16            var newTop = randInt(500,10);
17            var newLeft = randInt(700,10);
18            $(this).animate(
19              {top:newTop, left:newLeft}, 200); });
20      });
21    </script>
22    <style>
23      span {border:3px ridge blue; padding:5px;
➥position:fixed;
24        border-radius: 30px 10px; background-
➥color:blue; color:white;
25        font:bold italic 22px cursive, serif; }
26    </style>
27    </head>
28    <body>
29      <span>Click Here</span>
30    </body>
31 </html>
```

ch1205.html

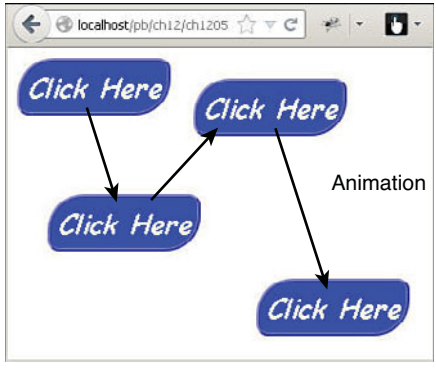


Figure 12.6 Animating moving and element using jQuery code in ch1205.html.

13

Using AJAX to Communicate with Web Servers and Web Services

AJAX communication is one of the most vital parts of most web sites. The sites allow jQuery and JavaScript to get additional data from the server and update individual elements on the page instead of reloading or loading a new web page.

The phrases in this chapter are designed to help you explore the world of asynchronous communication with the server using AJAX requests in jQuery and JavaScript.

Understanding AJAX

Despite its importance, AJAX tends to be a bit daunting to get into at first. With all the communication terms, it may seem easy to get overwhelmed. That really shouldn't be the case. AJAX is simply a request from

jQuery or JavaScript to the web server. The request sends data to the server, and the server responds with a success or failure and possibly additional data. That is it—nothing more, nothing less.

Watch out!

Don't confuse server-side dynamic creation of web pages with AJAX. Dynamic creation of web pages is still the old traditional method, but it is just a bit easier to manage. Each request back to the server still requires a full new web page to be returned. The only advantage is that the web page is generated in memory instead of read from disk.

The cool thing about AJAX requests is that they only send and receive the bits of data that are necessary instead of full pages. Traditional web requests always build full web pages that are loaded/reloaded in the browser. Figure 13.1 illustrates the difference between the two methods of updating data in the browser. The following is a list of a few of the benefits of AJAX requests:

- Less data requested from the web server.
- Allows the user to continue using the web page even while the request is being processed.
- Errors can be caught and handled on the client side before a request is ever made to the server.

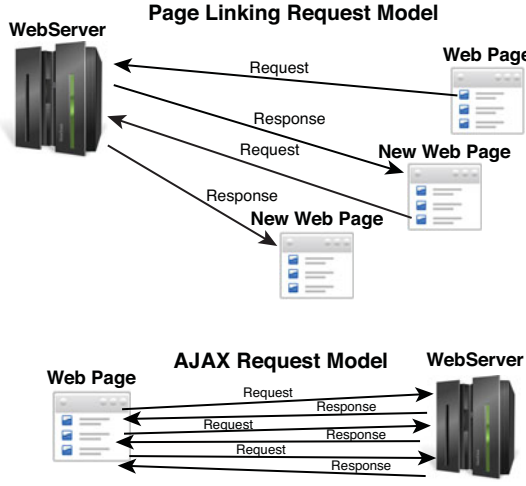


Figure 13.1 Comparison of AJAX requests to traditional page linking.

Understanding Asynchronous Communication

When you request a web page from the web server, the communication is synchronous to your browser. That means the browser waits until the HTML document is downloaded from the server before it begins rendering it and retrieving the additional resources necessary.

Did you know?

You can do synchronous AJAX requests with both jQuery and JavaScript. The jQuery `.ajax()` method and the JavaScript `send()` function both have a boolean field that allows this. I don't recommend this, though, because you can cause the browser to lock up, which creates some unhappy users. In fact, in jQuery 1.8 and later, that option is deprecated.

Asynchronous AJAX communication is different. When you send an AJAX request, control is returned immediately back to the jQuery or JavaScript, which can do additional things. Only when the request has completed or timed out are events triggered in the browser that allow you to handle the request data.

Understanding Cross-Domain Requests

Cross-domain requests occur when you send AJAX requests to separate servers from different domains. The browser prevents this, and correctly so, because of a multitude of security reasons.

The only problem with blocking cross-domain requests is that you often want to obtain data from services external to the current web site. You can get around this in a couple of different ways.

The first method is to have the web server act as a proxy to the other servers or services, meaning that instead of directly communicating via JavaScript, you send the request to the server and have the server do it for you.

Another option is to do what is called *on-demand JavaScript*, which JSON with Padding (JSONP) uses. This method takes advantage of the fact that you can download a script from another web site as a resource using the following syntax:

```
<script type="text/javascript"
        src="http://new.domain.com/getData?jsonp=
➥parseData">
</script>
```

Understanding GET versus POST Requests

The two main types of requests that you send to the web server are GET and POST. A GET request passes parameters as part of the URL, whereas a POST request passes them as part of the request data.

GET Request URL:

```
http://localhost/code/example1.html?first=Brad&last=
➥Dayley
```

GET Request Data:

```
<empty>
```

POST Request URL:

```
http://localhost/code/example1.html
```

POST Request Data:

```
first=Brad
last=Dayley
```

Understanding Response Data

When you send an AJAX request to the server, the server responds with response data. There are four main types of response that you will be working. These response data types are script, text, JSON, and XML/HTML.

In JavaScript, the response data is available in the `response` and `responseText` attributes of the `XMLHttpRequest` object once the request completes. In jQuery, the response data is passed as the first parameter to the AJAX handler function you define.

The script and text are handled pretty simply by the `.load()` and `.getScript()` methods. JSON and XML/HTML can be a bit more complex.

Using `.getScript()` to Load JavaScript Using AJAX

```
$.getScript("scripts/ajax/newLibray.js",
  function(script, status, jqxr){
    . . . handler code here . . .
  });
```

Script data is JavaScript code that can be loaded in the browser and subsequently used after the AJAX request. The best way to handle script data is to use the jQuery `.getScript(url [,successHandler(script,status, jqXHR))` method. This method loads the script automatically so that it is available for use.

The `url` is the location of the script on the server, and the optional `successHandler` is the AJAX handler function that is triggered on success.

Handling Text Data

```
. . . jQuery . . .
function ajaxHandler(data){
  $("#myP").html(data); }
. . . JavaScript . . .
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function() {
  document.getElementById(
    "myP").innerHTML=xmlhttp.responseText;
}
```

Text data is the most basic type of return value. You can treat the text data as a simple string. For example, in jQuery, to set a paragraph element to the text returned by the server, you would use the following AJAX handler:

```
function ajaxHandler(data){
  $("#myP").html(data); }
```

In JavaScript, you can just use the `responseText` attribute of the `XMLHttpRequest` object:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function() {
  document.getElementById(
    "myP").innerHTML=xmlhttp.responseText;
}
```

Using JSON Response Objects

```
function ajaxHandler(jsonData){
  var name  = jsonData.first + " " + jsonData.last;
}
```

JSON data is by far the easiest format to work with in AJAX responses when dealing with complex data. In

jQuery, JSON data is automatically converted to a JavaScript object. So you can access the data via dot naming. For example, the following JSON response from the server:

```
{"first":"Brad", "last":"Dayley"}
```

Can be accessed in the response data as follows:

```
var name  = data.first + " " + data.last;
```

Converting JSON Response Strings to JavaScript Objects

```
. . . jQuery . . .
var data = $.parseJSON(
  '{"first":"Brad", "last":"Dayley"}');
var name  = data.first + " " + data.last;
. . . JavaScript . . .
var data = JSON.parse(
  '{"first":"Brad", "last":"Dayley"}');
var name  = data.first + " " + data.last;
```

In JavaScript and even some cases in jQuery, you end up with a text response that contains a JSON string. That is also simple to handle. In jQuery, you can use `$.parseJSON()` to convert the JSON string to a JavaScript object. For example:

```
var data = $.parseJSON('{"first":"Brad",
➥"last":"Dayley"}');
var name  = data.first + " " + data.last;
```

In JavaScript, you use the `JSON.parse()` method to convert the JSON string to a JavaScript object:

```
var data = JSON.parse('{"first":"Brad",
➥"last":"Dayley"}');
var name  = data.first + " " + data.last;
```

Using XML/HTML Response Data Response Objects

```
. . . jQuery . . .
var name  = $(data).find("first").text() + " " +
   $(data).find("last").text();;
}
. . . JavaScript . . .
var first =
  xmlhttp.responseXML.getElementsByTagName("first");
var last =
  xmlhttp.responseXML.getElementsByTagName("last");
```

XML/HTML data is not as easy as JSON, but jQuery does make it fairly simple to work with. XML data in the response is returned as a DOM object, which can be converted to jQuery and searched/navigated using jQuery's extensive options. For example, the following XML response from the server:

```
<person><first>Brad</first><last>Dayley</last></
➥person>
```

Can be accessed in the response data as follows:

```
var name  = $(data).find("first").text() + " " +
➥$(data).find("last").text();
```

The XML/HTML data can be accessed in the JavaScript AJAX response in the `responseXML` attribute of the `XMLHttpRequest` object. For example:

```
var first =
➥xmlhttp.responseXML.getElementsByTagName("first");
var last =
➥xmlhttp.responseXML.getElementsByTagName("last");
```

Converting XML/HTML Response Data into XML DOM Object

```
var xmlStr =
  "<person><first>Brad</first><last>Dayley</last></
➥person>");
. . . jQuery . . .
var data = $.parseXML(xmlStr);
var name  = $(data).find("first").text() + " " +
   $(data).find("last").text();
. . . JavaScript . . .
parser=new DOMParser();
xmlDoc=parser.parseFromString( xmlStr,"text/xml");
var last = xmlDoc.getElementsByTagName("last");
```

Similar to JSON, if the response data object comes as a string, you can use the `.parseXML()` to get a DOM object. For example:

```
var data = $.parseXML(
  "<person><first>Brad</first><last>Dayley</last></
➥person>");
var name  = $(data).find("first").text() + " " +
   $(data).find("last").text();
```

In JavaScript, you need to get an XML `DOMParser` object and then call the `parseFromString(xmlString, "text/xml")` method. This returns a DOM object of the XML string:

```
parser=new DOMParser();
xmlDoc=parser.parseFromString(
  xmlhttp.responseText,"text/xml");
```

AJAX from JavaScript

To implement an AJAX request in JavaScript, you need access to a new `window.XMLHttpRequest` object. Table 13.1 describes the important methods and attributes of the `XMLHttpRequest` object.

Table 13.1 **Important Methods and Attributes of the `XMLHttpRequest` Object**

Method/Attribute	Description
`open()`	Allows you to construct a `GET` or `POST` request.
`send()`	Sends the request to the server.
`onreadystatechange`	Event handler that is executed when the state of the `XMLHttpRequest` object changes.
`response`	Object created by the browser based on the data type.
`responseText`	Raw text returned in the response data from the server.
`setRequestHeader`	Allows you to set HTTP request headers that are necessary to implement the request.
`status`	Status response code from server (for example, `200` for success, `404` for not found, and so on).

Geting an XMLHttpRequest Object

```
var xmlhttp;
if (window.XMLHttpRequest){ //newer Browsers
  xmlhttp=new XMLHttpRequest(); }
else {// Older IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); }
```

Newer browsers provide the `XMLHttpRequest` object directly as `window.XMLHtpRequest`. Older IE browsers do not provide an `XMLHttpRequest` object directly. Instead, they use an `ActiveXObject("Microsoft.XHMLHTTP")` object. If you want your AJAX code to support the older browsers, you need to add an alternative way of getting the access to an `XMLHttpRequest` by building the IE version if `window.XMLHttpRequest` is not available.

Sending a GET Request from JavaScript

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { document.getElementById("email").innerHTML=
➥xmlhttp.responseText;}}
xmlhttp.open("GET","getUserEmail.php?userid=brad",
➥true);
xmlhttp.send();
```

To send a `GET` request from JavaScript, you first need to implement an `onreadystatechange` event handler to handle the `response` or `responseText` values. Then you need to call `open("GET", URL)` to open the connection and `send()` to send the request.

The following code shows an example of a `GET` request that is sent to the web server via JavaScript. The `user`

value is added to the query string in line 15. Figure 13.2 shows the response string below the input field:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script>
06     var xmlhttp = new XMLHttpRequest();
07     xmlhttp.onreadystatechange=function() {
08       if (xmlhttp.readyState==4 && xmlhttp.
➥status==200){
09         document.getElementById(
10
➥"message").innerHTML=xmlhttp.responseText;
11       }
12     }
13     function getEmail(){
14       var user =
➥document.getElementById("email").value;
15
➥xmlhttp.open("GET","getEmail.php?user="+user);
16       xmlhttp.send();
17     }
18   </script>
19   </head>
20   <body>
21     <input id="email" type="text" />
22     <button onclick="getEmail()">Get Email</
➥button>
23     <p id="message"></p>
24   </body>
25 </html>
```

ch1301.html

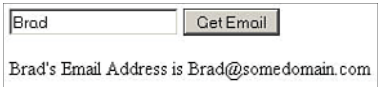


Figure 13.2 Results of sending a GET request using JavaScript code in ch1301.html.

Sending a POST Request from JavaScript

```
var xmlhttp = new XMLHttpRequest();
var params = "first=Brad&last=Dayley&email=brad@
➥dayleycreations.com";
xmlhttp.setRequestHeader("Content-type",
  "application/x-www-form-urlencoded");
xmlhttp.setRequestHeader("Content-length",
➥params.length);
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { alert("Email Updated: " +
➥xmlhttp.responseText); }
}
xmlhttp.open("POST","setUserEmail.php",true);
xmlhttp.send(params);
```

To send a POST request from JavaScript, you first need to implement an onreadystatechange event handler to handle the response or responseText values. For example:

```
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { alert("Email Updated: " +
➥xmlhttp.responseText); }
}
```

Next, you need to add the Content-length and Content-type headers to make sure that the data is treated correctly at the server. The Content-length is set

to the length of the `params` string. The `Content-type` tells the server how to handle the data in the `POST`. Use `"application/x-www-form-urlencoded"` when submitting form data.

```
var params = "first=Brad&last=Dayley&email=brad@
➥dayleycreations.com";
http.setRequestHeader("Content-type",
  "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length",
➥params.length);
```

Finally, you can call `open("POST", URL)` to open the connection and `send(params)` to send the request:

```
xmlhttp.open("POST","setUserEmail.php",true);
xmlhttp.send(params);
```

The following code shows an example of a `POST` request that is sent to the web server via JavaScript. The `user` and `email` values are added to the `params` string that is sent in the `send(params)` call in line 24. Figure 13.3 shows the response string below the input field:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script>
06     var xmlhttp = new XMLHttpRequest();
07     xmlhttp.onreadystatechange=function() {
08       if (xmlhttp.readyState==4 && xmlhttp.
➥status==200){
09         var response =
```

```
➥JSON.parse(xmlhttp.response);
10          document.getElementById(
11            "message").innerHTML=response.msg;
12        }
13      }
14      function setEmail(){
15        var params = "user="
16        params +=
➥document.getElementById("user").value;
17        params += "&email=";
18        params +=
➥document.getElementById("email").value;
19        xmlhttp.open("POST","postEmail.php",true);
20        xmlhttp.setRequestHeader("Content-type",
21          "application/x-www-form-urlencoded");
22        xmlhttp.setRequestHeader("Content-length",
23          params.length);
24        xmlhttp.send(params);
25      }
26    </script>
27    </head>
28    <body>
29      <input id="user" type="text">User</input><br>
30      <input id="email"
➥type="text">Email</input><br>
31      <button onclick="setEmail()">Set Email</
➥button>
32      <p id="message"></p>
33    </body>
34 </html>
```

ch1302.html

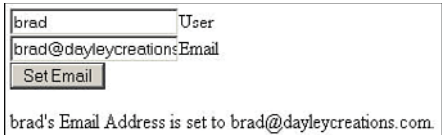


Figure 13.3 Results of sending a POST request using JavaScript code in ch1302.html.

AJAX from jQuery

jQuery provides wrapper functions that offer a bit cleaner interface to send AJAX requests. In jQuery, the `XMLHttpRequest` object is replaced by the `jqXHR` object, which is a jQuery version with a few additional methods to support the jQuery AJAX wrapper functions. The following is a list of the jQuery AJAX wrapper functions:

- **`.load(url [, data ] [, success(data, textStatus, jqXHR) ] )`**—This method is used to load data from the server directly into the elements represented by jQuery object.
- **`.getScript (url [, data ] [, success(data, textStatus, jqXHR) ] ) )`**—This method is used to load and then immediately execute a JavaScript/jQuery script.
- **`.getJSON(url [, data ] [, success(data, textStatus, jqXHR) ] ) )`** —This method is used to load data in JSON format using a JSONP request to servers on different domains.
- **`.get(url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ]) )`**—This method is used to send a generic `GET` request to the server.

- **`.post(url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ]) )`**—This method is used to send a generic `POST` request to the server.

Each of these methods allows you to specify the `url` of the request. The `.load()` method loads data directly into elements represented by the jQuery object. The `.get()` and `.post()` methods are used to send `GET` and `POST` requests.

The `data` argument can be a string or a basic JavaScript object. In the following example, `obj`, `objString`, and `formString` are all valid data arguments:

```
var obj ={"first":"Brad", "last":"Dayley"};
var objString = $.param(obj);
var formString = $("form").serialize();
```

You can also specify the function that executes when the response from the server succeeds. For example, the following success handler sets the value of the `#email` element to the value response data:

```
$.get("getEmail.php?first=Brad&last=Dayley", null,
➥function (data, status, xObj){
  $("#email").html(data);
}));
```

The `.get()` and `.post()` methods also allow you to specify a `dataType` parameter as a string, such as `"xml"`, `"json"`, `"script"` , and `"html"`, that formats the expected format of the response data from the server. For example, to specify a response type of JSON, you would use this:

```
$.get("getUser.php?first=Brad&last=Dayley", null,
➥function (data, status, xObj){
  $("#email").html(data.email);
}), "json");
```

Loading HTML Data into Page Elements

```
$("#content").load("new.html");
```

A great way to add dynamic data to a web page is to have the web server deliver the data already formatted as HTML. Then, using the jQuery `.load()` AJAX method, you can load the data directly into an element.

The `.load()` AJAX method can be called from any jQuery object and performs an AJAX get request in the background, formats the response, and injects it as the `innerHTML` of the element. For example:

```
$("#content").load("new.html");
```

The following code shows an example of using the `.load()` method to load lorem ipsum article data from the web server into an existing `<div>` container. Figure 13.4 illustrates loading the different articles by clicking on the nav items on the left:

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
```

```
08      function setArticle(){
09        $("#content").load($(this).attr
➥("article")+".html");
10      }
11      $(document).ready(function(){
12        $(".navItem").click(setArticle);
13      });
14    </script>
15    <style>
16      div {display:inline-block; float:left;
17        text-align:center; }
18      .navItem { border:1px dotted; display:block;
19        margin:3px; cursor:pointer;}
20      #content {border:1px solid blue;}
21      #article { width: 372px; height:360px;
➥padding:10px;
22        overflow-y:scroll; border:1px solid;}
23      #title { font-size:20px; display:block;
24        background-color:black; color:white; }
25      p {background-color:#DDDDDD; border-radius:
➥5px; }
26    </style>
27    </head>
28    <body>
29      <div>
30        <div id="leftNav">
31          <span class="navItem" article="article1">
32            jQUery Under the Hood</span>
33          <span class="navItem" article="article2">
34            jQuery Your New Best Friend</span>
35        </div>
36        <div id="content">
37          <span id="title"></span>
38          <div id="article"></div>
39        </div>
40      </div>
```

```
41    </body>
42 </html>
```

ch1303.html

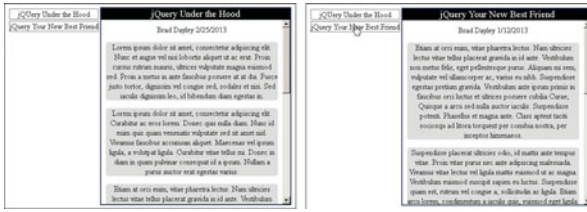


Figure 13.4 Loading HTML data directly into a `<div>` element using jQuery code in ch1303.html.

Using a jQuery `.get()` AJAX Request to Handle JSON Data

```
$.get("getJSONData.php", myHandler);
function myHandler(data){
  var responseValue = data.value;
}
```

Using jQuery to get JSON data from the server is extremely easy. Just call the `.get(url, handler(data))` method, pass in the `url` to the server location, and specify a `handler` to handle the data. For example:

```
$.get("getJSONData.php", myHandler);
```

The JSON data returned by the response will be a JSON object that is passed as the first argument to the handler function. You can access the data using dot syntax; for example:

```
function myHandler(data){
  var responseValue = data.value;
}
```

The best way to demonstrate how to handle a `.get()` request with JSON data is to show you an example. The following code makes a `.get()` request directly to a JSON file and then uses that data to build the image list shown in Figure 13.5.

```
<html>
  <head>
  <title>Python Phrasebook</title>
  <meta charset="utf-8" />
  <script type="text/javascript"
    src="../js/jquery-2.0.3.min.js"></script>
  <script>
    function updateImages(data){
      for (i=0; i<data.length; i++){
        var imageInfo =data[i];
        var img = $('<img />').attr("src",
          "../images/"+imageInfo.image);
        var title =
➥$("<p></p>").html(imageInfo.title);
        var div = $("<div></div>").append(img,
➥title);
        $("#images").append(div);
      }
    }
    $(document).ready(function(){
      $.get("images.json", updateImages);
    });
  </script>
  <style>
    div {border:3px ridge white; display:inline-
➥block;
```

```
24       box-shadow: 5px 5px 5px #888888;
➥margin:10px; }
25     p { background-color:#E5E5E5;
26       margin:0px; padding:3px; text-align:center;
➥}
27     img { height:130px; vertical-align:top;  }
28     #images { background-color:black;
➥padding:20px; }
29   </style>
30   </head>
31   <body>
32     <div id="images"></div>
33   </body>
34 </html>
```

ch1304.html

```
[ {"title":"Rugged Strength", "image":"bison.jpg"},
  {"title":"Great Heights", "image":"peak.jpg"},
  {"title":"Summer Fun", "image":"boy.jpg"},
  {"title":"Grandure of Nature",
➥"image":"falls.jpg"},
  {"title":"Soft Perfection", "image":"flower.jpg"},
  {"title":"Looking Forward", "image":"boy2.jpg"},
  {"title":"Joy of Finishing",
➥"image":"sunset.jpg"}]
```

JSON Server Response

Figure 13.5 Using a `.get()` AJAX request to turn JSON data into an image gallery using the jQuery code in ch1304.html.

Using a jQuery `.get()` AJAX Request to Handle XML Data

```
$.get("getXMLData.php", myHandler);
function myHandler(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var value = $(this).children("value").text();
  }
}
```

Using jQuery to get XML data from the server is a bit different from using JSON data. You still call the `.get(url, handler(data))` method and pass in the `url` to the server location and specify a `handler` to handle the data. For example:

```
$.get("getXMLData.php", myHandler);
```

However, the XML data returned by the response will be a DOM object that is passed as the first argument to the handler function. The best way to handle the

DOM object is to convert it to a jQuery object and then use the extensive jQuery filtering and navigating methods to get the data you want. For example:

```
function myHandler(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var value = $(this).children("value").text();
  }
}
```

The best way to demonstrate how to handle a `.get()` request with XML data is to show you an example. The following code makes a `.get()` request directly to an XML file and then uses that data to build the image list shown in Figure 13.6.

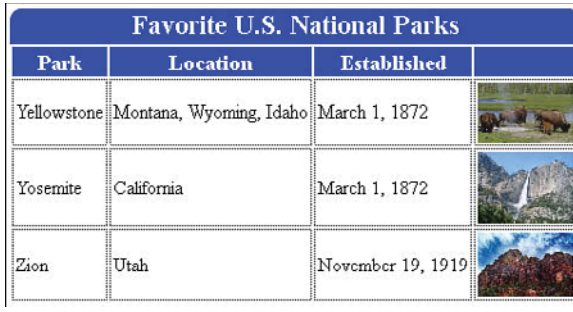
```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     function updateTable(data){
09       var parks = $(data).find("park");
10       parks.each(function(){
11         var tr = $("<tr></tr>");
12         tr.append($("<td></td>").html(
13           $(this).children("name").text()));
14         tr.append($("<td></td>").html(
15           $(this).children("location").text()));
16         tr.append($("<td></td>").html(
17
➥$(this).children("established").text()));
18         var img = $('<img />').attr("src",
```

```
➥"../images/"+
19            $(this).children("image").text());
20          tr.append($("<td></td>").append(img));
21          $("tbody").append(tr);
22        });
23      }
24      $(document).ready(function(){
25        $.get("parkdata.xml", updateTable);
26      });
27    </script>
28    <style>
29      img {width:80px;}
30      caption, th { background-color:blue;
➥color:white;
31        font:bold 18px arial black; }
32      caption { border-radius: 10px 10px 0px 0px;
33        font-size:22px; height:30px; }
34      td { border:1px dotted; padding:2px; }
35    </style>
36    </head>
37    <body>
38      <table>
39        <caption>Favorite U.S. National Parks</
➥caption>
40        <thead><th>Park</th><th>Location</th>
41
➥<th>Established</th><th> </th></thead>
42        <tbody></tbody>
43      </table>
44      <p></p>
45    </body>
46 </html>
```

ch1305.html

```
<parkinfo>
  <park>
    <name>Yellowstone</name>
    <location>Montana, Wyoming, Idaho</location>
    <established>March 1, 1872</established>
    <image>bison.jpg</image>
  </park>
  <park>
    <name>Yosemite</name>
    <location>California</location>
    <established>March 1, 1872</established>
    <image>falls.jpg</image>
  </park>
  <park>
    <name>Zion</name>
    <location>Utah</location>
    <established>November 19, 1919</established>
    <image>peak.jpg</image>
  </park>
</parkinfo>
```

XML Server Response



Favorite U.S. National Parks

Park	Location	Established	
Yellowstone	Montana, Wyoming, Idaho	March 1, 1872	
Yosemite	California	March 1, 1872	
Zion	Utah	November 19, 1919	

Figure 13.6 Using a `.get()` AJAX request to turn XML data into a table using the jQuery code in ch1305.html.

Using a jQuery .post() AJAX Request to Update Server Data

```
var params = [{name:"user",value:"Brad"},
              {name:"rating", value:"1"}]
 $.post("setRating.php", params, myHandler);
```

Using jQuery to post data to the server is easy with the `.post()` helper method because `.post()` handles all the necessary headers in the background. You call the `.post(url, params, handler(data))` method and pass in the `url` to the server location, parameter string, or object. Then you specify a `handler` to handle the data.

If you pass in an object, it needs to be a list of objects with `name` and `value` pairs. For example:

```
var params = [{name:"user",value:"Brad"},
              {name:"rating", value:"1"}]
 $.post("setRating.php", params, myHandler);
```

The handler function works the same way that the `.get()` handler function works. The following code shows an example of using a `.post()` request to update data on the server. The code you need to focus on is in lines 31–39. That function builds the `params` array and then sends the `.post()` request to the server to update the rating using a PHP script.

The PHP script alters the JSON file that is used to build the data shown in Figure 13.7. Notice that in the handler function for the `.post()` request retrieves the updated data from the server and updates the web page with the posted values.

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     function addTrip(data){
09       for (var idx in data){
10         var item = data[idx];
11         var div = $("<div></div>").attr("id",
12           item.location);
13         div.append($("<img></img>").attr("src",
14           "../images/"+item.image).addClass
➥("photo"));
15         div.append($("<span></span>").html(
16           item.location));
17         for(var i=0;i<parseInt(item.rating);i++){
18           div.append($("<img></img>").attr("src",
19           "../images/star.ico").addClass("star"))
➥; }
20         for(var i=0;i<(5-parseInt(item.rating));
➥i++){
21           div.append($("<img></img>").attr("src",
22           "../images/empty.ico").addClass
➥("star")); }
23         $("#vacations").append(div);
24       }
25       $(".star").click(sendRating);
26     }
27     function getTrips(){
28       $("#vacations").children().remove();
29       $.get("trips.json", addTrip);
30     }
31     function sendRating(){
32       var parent = $(this).parent();
33       var rating =
```

```
34          parent.children(".star").index($(this))
➥+1;
35        var params = [{name:"location",
36                       value:parent.attr("id")},
37                      {name:"rating", value:
➥rating}]
38        $.post("setRating.php", params, getTrips);
39      }
40      $(document).ready(function(){
41        getTrips();
42      });
43    </script>
44    <style>
45      #banner, #vacations { display:block;
46        font-size:30px; width:500px;}
47      div, span { display:inline-block;
48        width:120px; text-align:center;}
49      span { font-size:20px; }
50      .photo { height:50px;  border:5px ridge
➥white;
51        box-shadow: 10px 10px 5px #888888;
➥margin:20px;
52        border-radius:10px;}
53      .star { border:none; height:auto; width:auto;
➥}
54    </style>
55    </head>
56    <body>
57      <div><div id="banner">Vacations</div>
58      <div id="vacations"></div>
59    </body>
60 </html>
```

ch1306.html

```
location=Zion&rating=5
```

POST Data to setRating.php

```
[{"location":"Hawaii","image":"flower.jpg",
➥"rating":"4"},
{"location":"Yosemite","rating":"4","image":"falls.
➥jpg"},
{"location":"Montana","rating":"5","image":"bison.
➥jpg"},
{"location":"Zion","rating":"5","image":"peak.jpg"}]
```

JSON Server Response

Figure 13.7 A `.post()` AJAX request updates server data using the jQuery code in ch1306.html.

Handling jQuery AJAX Responses

```
$.get("getUser.php?first=Brad&last=Dayley", null,
➥function (data, status, xObj){
  alert("Request Succeeded");
}), "json").fail(function(data, status, xObj){
  alert("Request Failed");
}).always(function(data, status, xObj){
  alert("Request Finished");
});
```

In addition to specifying the success option, the wrapper methods also allow you to attach additional handlers using the following methods:

- **`.done(data, textStatus, jqXHR)`**—Called when a successful response is received from the server.
- **`.fail(data, textStatus, jqXHR)`**—Called when a failure response is received from the server or the request times out.
- **`.always(data, textStatus, jqXHR)`**—Always called when a response is received from the server.

For example, the following code adds an event handler to be called when the request fails:

```
$.get("getUser.php?first=Brad&last=Dayley", null,
➥function (data, status, xObj){
  $("#email").html(data.email);
}), "json").fail(function(data, status, xObj){
  alert("Request Failed");
});
```

To illustrate this in a practical example, the following code implements the `.done()`, `.fail()`, and `.always()` AJAX event handlers on a basic login request. The

login page sends the username and password to a server-side PHP script that checks for `username="user"` and `password="password"`. If the correct username and password are entered, the request succeeds; otherwise, the request fails. Alerts are called for all three types of handler functions, as shown in Figure 13.8.

```
01 <html>
02   <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <script type="text/javascript"
06     src="../js/jquery-2.0.3.min.js"></script>
07   <script>
08     function failure(){ alert("Login Failed"); }
09     function success(){ alert("Login Succeeded");
➥}
10     function always(){ alert("Login Request
➥done"); }
11     function login(){
12       $.get("login.php",
13        $("#loginForm").serialize()).done(
14          success).fail(failure).always(always);
15       return false;
16     }
17     $(document).ready(function(){
18       $("#loginButton").click(login);
19     });
20   </script>
21   <style>
22     #login { border-radius: 15px; text-align:
➥center;
23       height:180px; width:250px; border:3px ridge
➥blue;}
24     #title { background-color:blue; color:white;
25       border-radius:10px 10px 0px 0px;
➥height:30px;
```

```
26        font:bold 22px arial black; }
27      input { border-radius:10px; border:3px groove
➥blue;
28        margin-top:20px; padding-left:10px; }
29      label { font:italic 18px arial black; }
30    </style>
31    </head>
32    <body>
33      <div id="login">
34        <div id="title">Login</div>
35        <form id="loginForm">
36          <label>Username: </label>
37          <input type="text" name="user" /><br>
38          <label>Password: </label>
39          <input type="password" name="pw" /><br>
40          <input id="loginButton" type="button"
41                 value="Login" />
42        </form>
43      </div>
44    </body>
45 </html>
```

ch1307.html

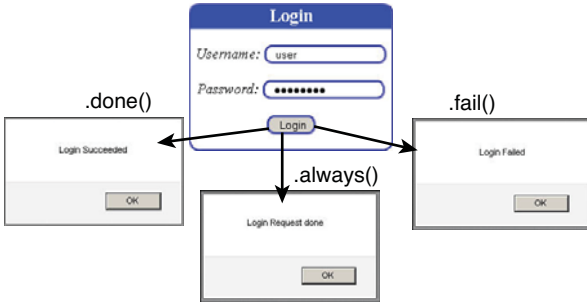


Figure 13.8 Handling success, failure, and always AJAX events using the jQuery code in ch1307.html.

Using Advanced jQuery AJAX

The concepts already covered in this chapter should take care of most of your AJAX needs. However, they far from cover the full power of the jQuery AJAX interface. The following sections discuss some of the additional AJAX functionality built directly into jQuery.

Changing the Global Setup

```
$.ajaxSetup({url:"service.php", accepts:"json})
```

jQuery provides the `.ajaxSetup()` method that allows you to specify options that configure AJAX requests throughout the script. Table 13.2 lists some of the options that can be specified when calling `.ajaxSetup()`. For example, the following code sets the default global URL for requests:

```
$.ajaxSetup({url:"service.php", accepts:"json})
```

Using Global Event Handlers

```
$(document).ajaxStart(function(){
  $("form").addClass("processing");
});
$(document).ajaxComplete(function(){
  $("form").removeClass("processing");
});
```

jQuery provides methods to create global event handlers that are called on events such as initialization or completion for all AJAX requests. The global events are fired on each AJAX request. Table 13.2 lists the methods that can be used to register global event handlers.

An example of using a global event handler to set the class of a form is shown here:

```
$(document).ajaxStart(function(){
  $("form").addClass("processing");
});
$(document).ajaxComplete(function(){
  $("form").removeClass("processing");
});
```

By the way

Global events are never fired for cross-domain script or JSONP requests, regardless of the value of global.

Table 13.2 **jQuery Global AJAX Event Handler Registration Methods**

Method	Description
`.ajaxComplete(function)`	Registers a handler to be called when AJAX requests are fully complete.
`.ajaxError(function)`	Registers a handler to be called when AJAX requests fail.
`.ajaxSend(function)`	Registers a function to be executed before an AJAX request is sent.
`.ajaxStart(function)`	Registers a handler to be called when the first AJAX request starts.
`.ajaxStop(function)`	Registers a handler to be called when all AJAX requests have completed.
`.ajaxSuccess(function)`	Registers a function to be executed whenever an AJAX request completes successfully.

Implementing Low-Level AJAX Requests

```
$.ajax({
  url:"setEmail",
  type:"get",
  accepts:"json",
  contentType: 'application/x-www-form-urlencoded;
➥charset=UTF-8',
  data: {"first":"Brad", "last":"Dayley"}
}).fail(function(){ alert("request Failed"); });
```

All AJAX request wrapper methods that you have been working with in his chapter are handled underneath through the `.ajax(url [, settings])` interface. This interface is a bit more difficult to use, but it gives you much more flexibility and control of the AJAX request.

The `.ajax()` method is called the same way that `.get()` and `.post()` are, but the settings argument allows you to set a wide variety of settings when making the request.

Table 13.3 lists the more common settings you can apply to the `.ajax()` method.

Table 13.3 **Common Settings Available in the `.ajax()` Request Method**

Method/ Attribute	Description
`accepts`	Specifies the content type(s) the server can send back in the response, such as application/Json.
`async`	Boolean. Default is `true`, meaning the request is sent and then received asynchronously.

Table 13.3 Continued

Method/ Attribute	Description
`beforeSend`	Specifies a function that should be run before sending the request.
`complete`	Function to execute when the response is fully received.
`contentType`	Sets the content type to send with the request. The server uses the content type to determine how to parse the data.
`crossDomain`	Boolean. Allows you to send a cross-domain request such as JSONP.
`data`	Specifies the data payload to send with the request.
`error`	Specifies a function to execute if the request fails.
`headers`	Object that contains the headers with values that should be sent to the server. For example: `{"Content-length": data.length}`
`success`	Function to execute if the response comes back with a `200` status, meaning the request was successful.
`timeout`	Specifies the milliseconds to wait before giving up on getting a response.
`type`	Set to `GET` or `PUT` to specify request type.
`url`	Specifies the URL on the web server to send the request.

A simple example of using the .ajax() interface is shown in the following code:

```
$.ajax({
  url:"setEmail",
  type:"get",
  accepts:"json",
  contentType: 'application/x-www-form-urlencoded;
➥charset=UTF-8',
  data: {"first":"Brad", "last":"Dayley"}
}).fail(function(){ alert("request Failed"); });
```

The .ajax() method returns a jqXHR method that provides some additional functionality, especially when handling the response. Table 13.4 lists some of the methods and attributes attached to the jqXHR object.

Table 13.4 **Common Methods and Attributes of the jqXHR Object Returned by .ajax()**

Method/Attribute	Description
abort()	Aborts the request.
always(function)	Specifies a function to be called when the request completes.
done(function)	Specifies a function to be called when the request completes successfully.
fail(function)	Specifies a function to be called when the request fails.
getAllResponseHeaders()	Returns the headers included in the response.
getResponseHeader(name)	Returns the value of a specific response header.

Table 13.4 Continued

Method/Attribute	Description
`setRequestHeader (name, value)`	Sets the value of an HTTP header to be sent with the AJAX request.
`readyState`	Values: 1—Has not started loading yet 2—Is loading 3—Has loaded enough, and the user can interact with it 4—Fully loaded
`status`	Contains the response status code returned from the server, such as `200` status, meaning the request was successful.
`statusText`	Contains the status string returned from the server.

14

Implementing Mobile Web Sites with jQuery

Mobile devices are the fastest growing development platform. Much of that development is geared toward making web sites mobile friendly. Users expect the portability provided by their mobile devices but the robustness that is already implemented in the traditional web sites.

This chapter focuses on using the jQuery Mobile library to help you get moving on creating some clean, cool mobile web pages. The jQuery Mobile library has extensive capabilities. The phrases in this chapter cover the basic structure, syntax, and concepts to help you implement a much wider variety of mobile solutions.

Getting Started with jQuery Mobile

Much of the jQuery Mobile framework is built on extending the current HTML elements with additional

functionality. jQuery Mobile builds on the HTML framework by adding `data` attributes to the HTML elements.

All the data attributes begin with `data-` and end with some slightly descriptive word of what the attribute means. The jQuery Mobile library's additional methods, events, styles, and values that fit the mobile needs are attached to the element. You will see this extensively in the examples to come.

Table 14.1 lists many of the attributes along with some of the values they support. The table is a good way to visualize how the data attribute method works and can act as a reference.

Table 14.1 **A Few of the Data Attributes Added in jQuery Mobile**

Event	Description
`data-role`	This is used to define the role that HTML will play in the mobile page, such as a dialog box, button, or panel.
`data-mini`	Each UI element has a compact version that is used if this is set to `true`.
`data-theme`	Specifies the letter (a–z) that is used when rendering the UI component.
`data-transition`	Specifies a transition effect to use when transitioning from one page/state to another, such as `slide` or `pop`.
`data-direction`	Specifies the direction of the animation so that it can match the transition. An example is reverse for page back.

Event	Description
data-rel	Defines the relationship that the element has to a link. Values include back, dialog, external, and popup.
data-title	Text displayed when page is shown.
data-icon	Specifies an icon to attach to an HTML element. Possible values are home, delete, plus, arrow-u, arrow-d, check, gear, grid, star, custom, arrow-r, arrow-l, minus, refresh, forward, back, alert, info, and search.
data-iconpos	Specifies the icon position when attached to an element: left, right, top, bottom, notext.
data-add-back-btn	For items with data-role="dialog", if true a back button is added.
data-collapsed	For items with role="collapsible", specifies whether the state is collapsed.
data-collapsed-icon	Defines the collapsed icon for role="collapsible" elements.
data-expanded-icon	Defines the expanded icon for role="collapsible" elements.
data-close-btn	For items with role="dialog", if true a close button is added.

Determining if the Page Is Being Viewed on a Mobile Device

```
if(
➥/Android|webOS|iPhone|iPad|iPod|BlackBerry/i.test(
  navigator.userAgent) ) {
   . . . mobile code here . . .
 }
```

You can use the following JavaScript and regex statement to parse the `navigator.userAgent` value and determine whether a user is coming into your web site on a mobile device:

```
if( /Android|webOS|iPhone|iPad|iPod|BlackBerry/
➥i.test(navigator.userAgent) ) { }
```

The statement parses the `userAgent` attribute and tries to find the most common strings incorporated in the mobile browsers.

Detecting Mobile Screen Size

```
$("#contentBody").width(screen.width).height(screen.
➥height);
```

In today's mobile world, the fact that a user is on a mobile device is no longer the critical question. The new critical question is this: How much screen space do users have to work? You need to check on the device's screen size. Does it have a 3-inch, 4-inch, or 11-inch screen? There is a big difference. To get the screen size, use `screen.height` and `screen.width` in JavaScript and then adjust your pages dynamically to support that size.

Specifying Different Theme Swatches

```
<div data-role="header" data-theme="b">
  <h1>Teach</h1></div>
<a data-role="button"
  data-theme="a" src="next">Next</a>
```

The CSS files that come with jQuery Mobile include several versions of styling called swatches. A *swatch* is just a letter assignment that specifies what color scheme to use when rendering the mobile elements. Swatches are typically specified using the `data-theme` attribute. For example:

```
<div data-role="header" data-
➥theme="b"><h1>Teach</h1></div>
```

Handling New Mobile Events

```
$("#myImage").on("swipeleft", function(e) {
  . . .swipe left code . . . });
$(document).on("pageLoad", function(e, data) {
  . . . page load code . . . });
$(document).on("pagebeforechange", function(e, data)
➥{
  var toPage = data.toPage;
  . . . page change code . . . });
```

One of my favorite features of jQuery Mobile is how easy it is to implement mobile events. The library automatically creates the events and adds them to elements based on the data attributes. If you want to add or remove a specific mobile event, you can do so using the `.on()` method from jQuery that you are already familiar with.

For example, to add a tap handler to an element `#myImage`, you would use the following:

```
$("#myImage").on("swipeleft", function() { });
```

Table 14.2 lists some of the mobile events and describes their purpose.

Table 14.2 **New Events Added by jQuery Mobile**

Event	Description
`tap`	Triggered on a quick touch.
`taphold`	Triggered after the touch is held for more than a threshold. Default is 750ms.
`swipe`	Triggered by a quick horizontal drag.
`swipeleft`	Triggered by a quick horizontal drag to the left.
`swiperight`	Triggered by a quick horizontal drag to the right.
`orientationchange`	Triggered when the device orientation changes from portrait to landscape or vice versa.
`scrollstart`	Triggered when scrolling starts.
`scrollstop`	Triggered when scrolling stops.
`pagebeforeload`	Triggered right before a mobile page is loaded into the DOM.
`pageload`	Triggered after a mobile page is loaded into the DOM.
`pagebeforechange`	Triggered right before a mobile page is changed to a different page. The two pages may be in the same HTML document.
`pagechange`	Triggered after a mobile page is changed to a different page.
`pagebeforeshow`	Triggered right before a page transition occurs. The event is triggered on the new page.
`pageshow`	Triggered after a page transition has occurred.

An important feature included with jQuery mobile is the ability to trigger and handle events linked to changing and loading pages.

When you change pages, the following events are triggered:

- `pagebeforechange`, `pagechange`, `pagebeforeload`, `pageload`, `pageshow`, `pagehide`

When you load pages, the following events are triggered:

- `pagebeforeload`, `pageload`

These events allow you to implement code to handle new pages being transitioned and prevent pages from being downloaded from the server. The events are implemented as standard jQuery events, and the object passed back to the handler includes things like `url`, `toPage`, `absUrl`, `dataUrl`, and `xhr` object as well as the options used for changing pages.

The following code shows an example of adding a `pageload` event handler:

```
$(document).on("pageload", function(e, obj){
  if($("#pageThree .ui-content").length) {
    $("#pageThree .ui-content").append("Page loaded
➥from ."+ obj.url); }
});
```

By the way

jQuery Mobile also includes several virtual mouse events, such as the mask, the mouse, and touch events, to allow developers to register just the basic mouse events. These work well for the most part, but there are still a few quirks with them. The virtual events are `vmouseover`, `vmouseout`, `vmousedown`, `vmousemove`, `vmouseup`, `vmouseclick`, and `vmousecancel`.

Changing Pages with jQuery Code

```
$("#pageTwo").on("swipeleft", function(){
  $.mobile.changePage("newPage.html", {transition:
➥"slide", reverse:true}); });
```

jQuery provides the `$.mobile.changePage(URL, options)` method to dynamically change pages, where `URL` is the new page location. Table 14.3 shows the available options for the `.changePage()` call. The following code is an example of adding a `swipeleft` event handler to load a remote web page when the user left-swipes the page on the device. Notice that a transition of "slide" is used, and reverse option is set to `true`:

```
$("#pageTwo").on("swipeleft", function(){
  $.mobile.changePage("newPage.html", {transition:
➥"slide", reverse:true}); });
```

Table 14.3 **Options for the `.changePage()` and `.loadPage()` Calls**

Method	Description
`changeHash`	Specifies whether the hash in the location bar should be updated.
`data`	Object or string data to send with an AJAX request.
`dataUrl`	The URL used when updating the browser location. If not specified, the value of the `data-url` attribute is used.
`pageContainer`	Specifies the element that should contain the page.
`reloadPage`	Boolean. Reloads the page from the server if it is already in the DOM.
`reverse`	Boolean. Specifies the direction the page change transition runs.
`role`	The `data-role` value to be used when displaying the page.
`showLoadMsg`	Boolean. Specifies whether the loading message should be shown.
`transition`	The transition to use when showing the page.
`type`	Specifies the method of the AJAX request: `"get"` (default) or `"post"`.

Loading Mobile Pages without Displaying Them

```
<$.mobile.loadPage("newPage.php",
  {data=$("form").serialize(), type="post"});
```

Another helpful function is `.loadPage(URL, options)`. It downloads the mobile page from the web server using an AJAX call but does not change the mobile page to

the downloaded one. Actually, `.changePage()` calls `.loadPage()` underneath to retrieve the page. Most of the options listed in Table 14.3 are also available via `.loadPage()`, except `changeHash`, `dataUrl`, `reverse`, and `transition`.

The `.loadPage()` function is useful for preloading pages in the initialization functions that you want available later but do not want to display yet. The following code shows an example of loading a page using `POST` data from a form:

```
$.mobile.loadPage("newPage.php",
  {data=$("form").serialize(), type="post"});
```

Defining the Viewport Meta Tag

```
<meta name="viewport" content="width=device-width,
  initial-scale=1">
```

A critical component of using jQuery Mobile is adding the viewport settings in a `<meta>` tag inside the page `<head>` tag. The viewport defines how the browser displays the page zoom level and the dimensions used.

Specifically, you need to set the `content="width=device-width, initial-scale=1"` as shown next. These settings force the device's browser to render the web page width at exactly the number of pixels available on the device:

```
<meta name="viewport" content="width=device-width,
  initial-scale=1">
```

Without specifying the viewport setting, the mobile page is at a much higher size than the screen width, making the page look small. The user can still zoom in on the web page.

Configuring jQuery Mobile Default Settings

```
<script src="../js/jquery-2.0.3.min.js"></script>
<script>
  $(document).bind( "mobileinit", function () {
    $.mobile.page.prototype.options.headerTheme =
➥"b";
    $.mobile.page.prototype.options.footerTheme =
➥"b";
  });
</script>
<script src="../js/jquery.mobile-
➥custom.min.js"></script>
```

jQuery Mobile is initialized when the library is loaded. Any page elements with jQuery Mobile tags are initialized as well and use the default settings to create mobile versions of the elements.

Occasionally, you may want to override the default settings. To do this, you need to add a `mobileinit` event handler to the document object before loading the jQuery Mobile library. Then you can add your default override code in that handler function.

I mostly use the `mobileinit` handler to change the default theme swatches because that can't be done on the fly after the library has been loaded. The following code shows an example of setting the default header and footer themes:

```
<script src="../js/jquery-2.0.3.min.js"></script>
<script>
  $(document).bind( "mobileinit", function () {
    $.mobile.page.prototype.options.headerTheme =
➥"b";
    $.mobile.page.prototype.options.footerTheme =
➥"b";
  });
</script>
<script src="../js/jquery.mobile-
➥custom.min.js"></script>
```

Notice that the init <script> is placed after jQuery is loaded but before jQuery Mobile is loaded.

Building Mobile Pages

Creating mobile pages is actually simple. Pages consist of elements that are enhanced in jQuery Mobile using the data tags discussed in the previous chapter. This section focuses on using the data tags to define mobile web pages.

Creating a Mobile Web Page

```
<div data-role="page">
  <div data-role="header"><h1>Header</h1></div>
  <div data-role="content" id="content">
    <p>Images</p>
    <img src="../images/flower.jpg" />
  </div>
  <div data-role="footer"><h4>Footer</h4></div>
</div>
```

Mobile pages are composed of three main parts: the header, the footer, and content between them. All three are not necessarily required, but it is a good idea to at

least have a header with the content, especially when working with multiple pages.

All these elements are defined by adding data-role attributes to elements. The content inside the elements can be just about anything, including text, images, forms, and lists.

The following code shows an example of creating a simple mobile web page with a header, footer, and text and image content. Figure 14.1 illustrates the look of the web page:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script
10     src="../js/jquery.mobile-
➥custom.min.js"></script>
11     <link rel="stylesheet"
12       href="../js/css/jquery.mobile-
➥custom.min.css" />
13   <script>
14     $(document).ready(function() {
15       checkForMobile();
16     });
17   </script>
18   <style>
19     p { text-align:center; font:italic 45px
➥Helvetica;
20         color:blue; margin:5px; }
```

```
21      img { width:235px; }
22    </style>
23  </head>
24  <body>
25    <div id="border"><div id="frame"><div data-
➥role="page">
26      <div data-role="header"><h1>Header</h1></div>
27      <div data-role="content" id="content">
28        <p>Yosemite</p>
29        <p>Falls</p>
30        <img src="../images/falls.jpg" />
31      </div>
32      <div data-role="footer"><h4>Footer</h4></div>
33    </div></div></div>
34  </body>
35  </html>
```

ch1401.html

Notice that each of the components has a distinct value for `data-role`. Figure 14.1 shows the resulting mobile web page.

By the way

All the mobile examples included in this chapter contain `<div id="border"><div id="frame">` elements in the HTML `<body>` and a call to `checkForMobile()` in the jQuery code. The `checkForMobile()` code is located in the included `initmobile.js` file I created. It styles those elements for non-mobile browsers so that a phone image is displayed as the background. I used this to capture the images for this chapter. I also use this concept with mobile development because I can see in my development browser. I then test on an actual device.

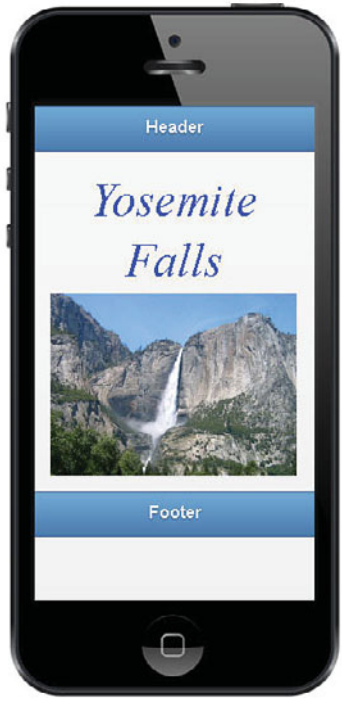


Figure 14.1 Basic mobile web page with a header, a footer, and a comment with text and images from ch1401.html.

Creating Fixed Headers and Footers

```
<div data-role="header" data-position="fixed">
  <h1>Header</h1></div>
...
<div data-role="footer" data-position="fixed">
  <h4>Footer</h4></div>
```

By default, both the header and the footer flow with the content, meaning that if you scroll the content up or down, the header scrolls with the content. Often, you want the header, footer, or both to stay in a fixed position on the device screen so that it is always displayed. jQuery Mobile makes that adjustment simple to make.

To make the header or footer—or a toolbar for that matter—fixed, all you have to do is add the `data-position="fixed"` attribute to the `<div>`. This make them stay in place, the header at the top of the page and the footer at the bottom regardless of how the content scrolls.

Implementing Mobile Sites with Multiple Pages

Mobile sites are composed of either a single HTML document with multiple `<div data-role="page">` elements or multiple HTML documents with those elements. Each `<div>` element represents a single mobile page.

When using multiple pages in your mobile web site, you need to implement code and UI controls to provide ways for the user to transition from one mobile page to another. The transitions should be smooth and intuitive based on the controls and content interaction.

You can change mobile pages by linking to the second page from the first using one of two methods: adding navigation buttons or programmatically changing the page in your jQuery code.

The pages can come from elements in the same web page or external URLs downloaded to the device. The following phrases describe the methods of implementing multiple page mobile sites.

Adding Navigation Buttons

```
<a href="#page2" data-role="button">Page2</a>
<a href="hour2201-page3.html" data-
role="button">Page</a>
```

Navigation buttons are links to other mobile pages. Typically, navigation buttons are added to the header or footer element for easy visibility. However, you can also place them inside the mobile content.

Navigation buttons are created by adding the data-role="button" attribute to an <a> link. The href attribute should point to the hash tag or URL of the mobile page you want to switch to. The following code shows the syntax for defining an <a> tag as a navigation link for a local link:

```
<a href="#page2" data-role="button">Page2</a>
```

The following code shows the syntax for defining an <a> tag as a navigation link for a remote link:

```
<a href="hour2201-page3.html"
➥data-role="button">Page3</a>
```

Positioning Navigation Buttons

```
<a href="#page3" data-role="button"
  class="ui-btn-right">Next</a>
<a href="#page1" data-role="button"
  class="ui-btn-left">Prev</a>
```

You can position the button on the left or right side of the header or footer, adding the `.ui-btn-right` or `.ui-btn-left` class to the `<a>`. This applies jQuery Mobile CSS styles to position the button on the right or left side of the header or footer.

Adding Page Change Transitions

```
. . . in HTML . . .
<a data-role="button" data-
➥transition="slide">Next</a>
<a data-role="button" data-transition="slide"
  data-direction="reverse">Prev</a>
. . . in jQuery . . .
$.mobile.changePage("#next", {transition:"slide" });
➥});
$.mobile.changePage("#prev", {transition:"slide",
  reverse:true }); });
```

Transitions are added to page changes by adding the data-transition attribute to the `<a>` link or other element that is generating the transition. You can also specify transition in the options element if you are calling `.changePage()`. For example:

```
<a data-role="button" data-
➥transition="slide">Next</a>
```

or

```
$.mobile.changePage("#next", {transition:"slide" });
➥});
```

You can also reverse the direction of the transition, which is especially useful if you want the screen to look like it is scrolling backward to a previous page.

To change the direction of the transition, use `data-direction="reverse"` in the `<a>` tag or add `reverse:true` to the `.changePage()` options. For example:

```
<a data-role="button" data-transition="slide"
  data-direction="reverse">Prev</a>
```

or

```
$.mobile.changePage("#prev", {transition:"slide",
  reverse:true }); });
```

Creating a Back Button

```
<a data-rel="back data-role="button"
  class="ui-btn-right">Next</a></div>
```

Another useful feature included in jQuery Mobile is the ability to define a navigation button as a back button. A back button uses the browser history to navigate to the previous button mobile page. To define a link as a back button, you need to add the `data-rel="back"` attribute. For example:

```
<a data-rel="back data-role="button" class="ui-btn-
➥right">Next</a></div>
```

Notice that there is not an `href` attribute. That is because the `href` attribute will be ignored; instead, the most recent URL will be popped off the browser's navigation history list.

Changing Pages with Swipe Event Handlers

```
$("#image").on("swipe", function(){
  . . . swipe code here . . .
});
$("#pageTwo").on("swipeleft", function(){
  $.mobile.changePage("#page3",
➥{transition:"slide"}); });
$("#pageTwo").on("swiperight", function(){
  $.mobile.changePage("#page1", {transition:"slide",
➥reverse:true}); });
```

The second method is to use the use the `$.mobile.changePage(URL, options)` function call, where the URL is the link location. Table 14.3 shows the available options for the `.changePage()` call.

The following is an example of adding a `swipeleft` event handler to load a remote web page when the user left-swipes the page on the device. Notice that a `transition` of `"slide"` is used, and the reverse option is set to `true` to make the transition slide to the left:

```
$("#pageTwo").on("swipeleft", function(){
  $.mobile.changePage("#page3", {transition:"slide",
➥reverse:true}); });
```

The following is an example of adding a `swiperight` event handler to load a remote web page when the user right-swipes the page on the device. Notice that a `transition` of `"slide"` is used, but the reverse option is not set:

```
$("#pageTwo").on("swiperight", function(){
  $.mobile.changePage("#page1", {transition:"slide"
➥}); });
```

Adding Navigation Buttons and Swipe Events Example

The following code shows a full example of implementing navigation buttons and swipe events to traverse three different pages. The code in lines 14–27 implement swipe event handlers that call `.changePage()` with transitions to handle finger swipes. Lines 38–47 implement the first mobile page, lines 48–59 implement the second mobile page, and lines 60–68 implement the third. Each of the pages has one or two navigation buttons in the header. Figure 14.2 illustrates the multipage app:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script
10     src="../js/jquery.mobile-
➥custom.min.js"></script>
11     <link rel="stylesheet"
12       href="../js/css/jquery.mobile-
➥custom.min.css" />
13   <script>
14     $(document).ready(function() {
15       checkForMobile();
16       $("#page1").on("swipeleft", function(){
17         $.mobile.changePage("#page2",
18           {transition:"slide"}); });
```

```
19        $("#page2").on("swiperight", function(){
20          $.mobile.changePage("#page1",
21            {transition:"slide", reverse:true});
➥});
22        $("#page2").on("swipeleft", function(){
23          $.mobile.changePage("#page3",
24            {transition:"slide", }); });
25        $("#page3").on("swiperight", function(){
26          $.mobile.changePage("#page1",
27            {transition:"slide", reverse:true});
➥});
28      });
29    </script>
30    <style>
31      p { text-align:center; font:italic 45px
➥Helvetica;
32          color:blue; margin:5px; }
33      img { width:235px; }
34    </style>
35  </head>
36  <body>
37    <div id="border"><div id="frame">
38    <div data-role="page" id="page1">
39      <div data-role="header"><h1>Page 1</h1>
40        <a data-role="button" href="#page2"
41          data-transition="slide"
42          class="ui-btn-right">Page 2</a>
43      </div>
44      <div data-role="content">
45        <p>Arches NP</p><img
➥src="../images/arch.jpg"/>
46      </div>
47    </div>
```

```
48   <div data-role="page" id="page2">
49     <div data-role="header"><h1>Page 2</h1>
50       <a data-role="button" href="#page1"
51         data-transition="slide"
52         data-direction="reverse">Page 1</a>
53       <a data-role="button" href="#page3"
54         data-transition="slide">Page 3</a>
55     </div>
56     <div data-role="content">
57       <p>TiKal</p><img src="../images/
➥pyramid2.jpg"/>
58     </div>
59   </div>
60   <div data-role="page" id="page3">
61     <div data-role="header"><h1>Page 3</h1>
62       <a data-role="button" href="#page2"
63         data-transition="slide"
64         data-direction="reverse">Page 2</a>
65     </div>
66     <div data-role="content">
67       <p>Sunset</p><img src="../images/jump.jpg"
➥/></div>
68   </div>
69   </div></div>
70 </body>
71 </html>
```

ch1402.html

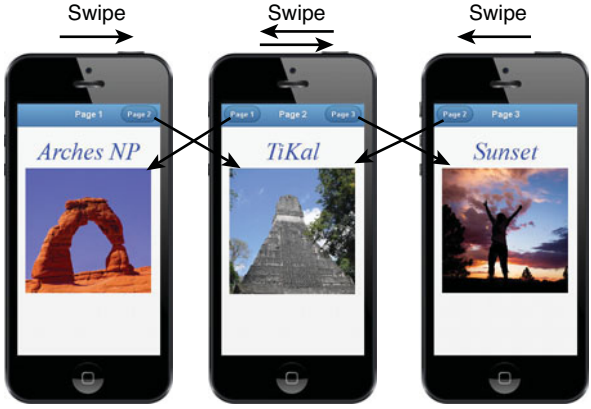


Figure 14.2 Multipage web site that allows you to navigate using navigation buttons or page swipes ch1402.html.

Creating a Navbar

```
<div data-role="navbar" >
  <ul>
    <li><a href="#pageTwo">Page 2</a></li>
    <li><a href="#pageThree">page 3</a></li>
    <li><a href="#pageFour">page 4</a></li>
  </ul>
</div>
```

Another method of navigating pages is a navbar. A *navbar* is a set of buttons grouped together in a single bar element. Each button links to a different mobile page.

Navbars are defined by adding the `role="navbar"` to a `<div>` and then adding a list of pages to link to using `<ul>`, `<li>`, and `<a>` elements. For example, the following code renders a navbar similar to the one in Figure 14.2:

```
<div data-role="navbar" >
  <ul>
    <li><a href="#pageTwo">Page 2</a></li>
    <li><a href="#pageThree">page 3</a></li>
    <li><a href="#pageFour">page 4</a></li>
  </ul>
</div>
```

The navbar `<div>` can be placed anywhere; you can put it in the header as shown in Figure 14.3, in the footer, with the content, or it can stand alone between the other sections of the mobile page. A common place to put the navbar is in a fixed footer. This allows the navbar to remain present even as the content scrolls.

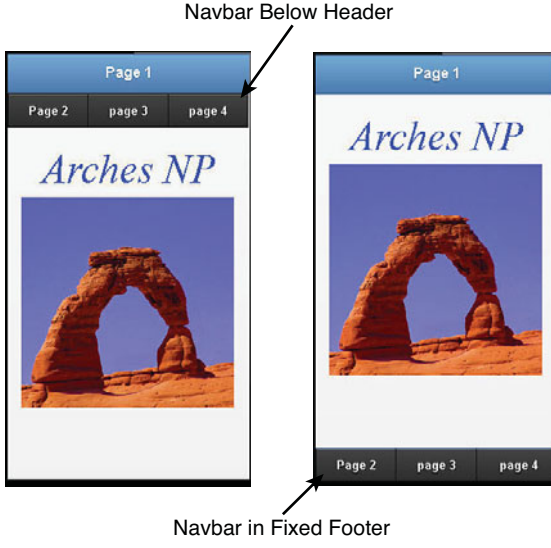


Figure 14.3 Mobile web page with navbar below the header and as a fixed footer.

Applying a Grid Layout

```
<div class="ui-grid-a">
  <div class="ui-block-a">Row 1 Column 1</div>
  <div class="ui-block-b">Row 1 Column 2</div>
  <div class="ui-block-a">Row 2 Column 1</div>
  <div class="ui-block-b">Row 2 Column 2</div>
</div>
```

One of the basic layouts provided by jQuery Mobile is the grid. The idea behind the grid layout is to split the page into equal-sized blocks, similar to an HTML table. You can then place content in those blocks, and they are automatically laid out correctly.

To add a grid layout, you need to add a `<div class="ui-grid-#">`, where `#` specifies the number of columns to include. The values are `a` (2 columns), `b` (3 columns), `c` (4 columns), and so on. For example, the following defines a three-column grid:

```
<div class="ui-grid-b"> </div>
```

Items are added to the grid by specifying a `<div ui-block-#>`, where `#` is the column letter, `a` (first), `b` (second), and so on. The first time a column letter is specified, the item is placed in row 1 of the grid in that column position. The second time it is placed in row 2. For example, to create a 2×2 grid, use the following code:

```
<div class="ui-grid-a">
  <div class="ui-block-a">Row 1 Column 1</div>
  <div class="ui-block-b">Row 1 Column 2</div>
  <div class="ui-block-a">Row 2 Column 1</div>
  <div class="ui-block-b">Row 2 Column 2</div>
</div>
```

You can also create a single-column grid using `<div class="ui-grid-solo">`. For example:

```
<div class="ui-grid-solo">
  <div class="ui-block-a"><p id="number"></p></div>
</div>
```

The best way to demonstrate how grid layouts work is to show you a practical example. The following code implements three different grid layouts in a single page to build a basic calculator app, shown in Figure 14.4. The first is a basic single-item grid with a number display, and then a 3×4 grid with calculator digit elements. The last grid is a 5×1 grid with calculator buttons:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script src="../js/jquery.mobile.js"></script>
10     <link rel="stylesheet"
11       href="../js/css/jquery.mobile.css" />
12   <script>
13     $(document).ready(function() {
14       $(document).ready(function() {
15         checkForMobile();
16         $("p").on("click", function(){
17           $("#number").append($(this).html());
➥});
```

```
18          $("span").on("click", function(){
19            if ($(this).html() === "="){
20              $("#number").html(eval($("#number").
➥html()));
21            } else {$("#number").html(""); };
22          });
23        });
24      });
25    </script>
26    <style>
27      p, span {  margin:2px; border-radius:15px;
28        background-color:#888888; color:white;}
29      p, span { font:bold 30px/50px arial;
30        text-align:center; border:3px ridge blue;}
31      #number { background-color:black; min-
➥height:50px;
32        text-align:right; padding-right:5px; }
33      span { background-color:#555555;
➥display:block; }
34      #logic span { background-color:#B10000; }
35      #logic p { background-color:#0066AA;  }
36    </style>
37 </head>
38 <body>
39    <div id="border"><div id="frame">
40      <div data-role="header"><h1>Grid
➥Page</h1></div>
41      <div data-role="content">
42        <div class="ui-grid-solo">
43          <div class="ui-block-a"><p id="
➥number"></p></div>
```

```
44        </div>
45        <div class="ui-grid-b">
46          <div class="ui-block-a"><p>1</p></div>
47          <div class="ui-block-b"><p>2</p></div>
48          <div class="ui-block-c"><p>3</p></div>
49          <div class="ui-block-a"><p>4</p></div>
50          <div class="ui-block-b"><p>5</p></div>
51          <div class="ui-block-c"><p>6</p></div>
52          <div class="ui-block-a"><p>7</p></div>
53          <div class="ui-block-b"><p>8</p></div>
54          <div class="ui-block-c"><p>9</p></div>
55          <div class="ui-block-
➥a"><span>C</span></div>
56          <div class="ui-block-b"><p>0</p></div>
57          <div class="ui-block-
➥c"><span>CE</span></div>
58        </div>
59        <div class="ui-grid-d" id="logic">
60          <div class="ui-block-a"><p>+</p></div>
61          <div class="ui-block-b"><p>-</p></div>
62          <div class="ui-block-c"><p>*</p></div>
63          <div class="ui-block-d"><p>/</p></div>
64          <div class="ui-block-
➥e"><span>=</span></div>
65        </div>
66      </div>
67    </div></div>
68 </body>
69 </html>
```

ch1403.html

Figure 14.4 Using the jQuery Mobile grid layout to implement a basic calculator app in ch1403.html.

Implementing Listviews

One of the most common ways to organize mobile content is with listviews. Listviews organize the content into scrollable, linkable lists that are easy to view and navigate. jQuery UI does a great job of providing a framework to easily implement listviews in your code.

The application of listviews varies a lot depending on the amount and type of data that is being placed in them. To handle this, jQuery provides an array of different types of lists. The following sections cover the most commonly used.

Watch out!

If you dynamically add items to lists, tables, and so on in jQuery code, you need to call the `refresh()` action on that element to refresh the contents with jQuery Mobile.

Creating a Basic Listview

```
<ul data-role="listview">
  <li><a href="#one">Link 1</a></li>
  <li><a href="#two">Link 2</a></li>
  <li>Non Linkable Item</li>
</ul>
```

Lists are created by adding the `data-role="listview"` to a `<ul>` or `<ol>` element. jQuery automatically handles formatting the `<li>` elements into a list form. For `<ol>` elements, the formatted listview contains the numbering for each line item.

Using Nested Lists

```
<ul data-role="listview" >
  <li>Hobbits<ul><li>Frodo</li><li>Sam</li>
    <li>Bilbo</li></ul></li>
  <li>Elves<ul><li>Legolas</li><li>Elrond</li>
    <li>Galadriel</li></ul></li>
  <li>Men<ul><li>Aragorn</li><li>Boromir</li>
    <li>Theoden</li></ul></li>
</ul>
```

Nested lists are created by nesting additional <ul> elements inside the listview. jQuery Mobile automatically detects this and builds up linkable pages to the sublists. The following code shows an example of implementing nested lists, as shown in Figure 14.5:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script src="../js/jquery.mobile.js"></script>
10     <link rel="stylesheet"
11       href="../js/css/jquery.mobile.css" />
12   <script>
13     $(document).ready(function() {
14       checkForMobile();
15     });
16   </script>
17 </head>
18 <body>
19   <div id="border"><div id="frame">
20   <div data-role="page" id="nested">
21     <div data-role="header"><h1>Nested
➥List</h1></div>
22     <div data-role="content">
23     <ul data-role="listview">
24       <li>Hobbits<ul><li>Frodo</li><li>Sam</li>
25         <li>Bilbo</li></ul></li>
26
➥<li>Elves<ul><li>Legolas</li><li>Elrond</li>
27         <li>Galadriel</li></ul></li>
```

```
28        <li>Men<ul><li>Aragorn</li><li>Boromir</li>
29          <li>Theoden</li></ul></li>
30      </ul></div></div>
31    </div></div>
32 </body>
33 </html>
```

ch1404.html

Figure 14.5 Applying nested lists using the jQuery Mobile code in ch1404.html.

Implementing Split Button List

```
<ul data-role="listview">
  <li><a href="#">Jeep</a><a href="#" data-
➥icon="star">Like</a></li>
  <li><a href="#">Ford</a><a href="#" data-
➥icon="star">Like</a></li>
  <li><a href="#">Chevy</a><a href="#" data-
➥icon="star">Like</a></li>
</ul>
```

Split-button lists are lists that include multiple options on each line. This can be useful in a variety of ways. When listing products for sale, the main part can link to more details and the secondary link can add the item to the cart. To create a split-button list, create two `<a>` elements in the `<li>` item.

Adding Dividers to Lists

```
<ul data-role="listview" >
  <li data-role="list-divider">Numbers</li>
    <li>1</li><li>2</li><li>3</li>
  <li data-role="list-divider">Letters</li>
    <li>A</li><li>B</li><li>C</li>
</ul>
. . .
<ul data-role="listview" data-autodividers="true">
  <li>Alex</li><li>Alice</li><li>Brad</li><li>DaNae
➥</li><li>David</li>
  <li>Isaac</li><li>Jordan</li><li>Nancy</li>
</ul>
```

A *divided list* is one in which elements of the list are divided from each other by a simple bar. The idea is to make it easier for the user to see the list items by splitting up the view.

You can create divided lists manually by injecting your own dividers and adding a `data-role="list-divider"` to one of the `<li>` elements. For example:

```
<ul data-role="listview" >
  <li data-role="list-divider">Numbers</li>
    <li>1</li><li>2</li><li>3</li>
  <li data-role="list-divider">Letters</li>
    <li>A</li><li>B</li><li>C</li>
</ul>
```

You can also automatically add dividers by adding `data-autodividers="true"` to the `<ul>` element. This splits the elements every time the first character changes and creates a divider for that letter. For example, the following code adds an autodivider:

```
<ul data-role="listview" data-autodividers="true">
  <li>Alex</li><li>Alice</li><li>Brad</li><li>DaNae
➥</li><li>David</li>
  <li>Isaac</li><li>Jordan</li><li>Nancy</li>
</ul>
```

Implementing a Searchable List

```
<ul data-role="listview" data-filter="true">
  <li>Rome</li><li>Milan</li><li>Florence</li><li>
➥Genoa</li><li>Venice</li>
  <li>Naple</li><li>Balonga</li><li>Bari</li><li>
➥Turin</li><li>Palermo</li>
</ul>
```

Another useful feature is the searchable list. jQuery Mobile has a nice search feature built in that allows you to search the current list. The search feature adds a

text input at the top of the list and filters the items as you type text into the list. Only the items that match the filter text are displayed.

Searchable lists are created by adding `data-filter="true"` to the `<ul>` element containing the list. For example, the following code adds a searchable list:

```
<ul data-role="listview" data-filter="true">
  <li>Rome</li><li>Milan</li><li>Florence</li><li>
➥Genoa</li><li>Venice</li>
  <li>Naple</li><li>Balonga</li><li>Bari</li><li>
➥Turin</li><li>Palermo</li>
</ul>
```

Using Collapsible Blocks and Sets

```
<div data-role="collapsible" data-collapsed="false">
   <h3>Photo Information</h3>
   <p>Single Content.</p>
</div>
<div data-role="collapsible-set">
  <div data-role="collapsible">
    <h3>Image 1</h3><img src"1.jpg" /></div>
  <div data-role="collapsible" >
    <h3> Image 2</h3><img src="2.jpg" /></div>
  <div data-role="collapsible">
    <h3> Image 3</h3><img src="3.jpg" /></div>
</div>
```

Another useful way to represent content is by dividing it into collapsible elements. A header is presented that the user can see, but the content the header represents is hidden until the header is clicked.

This allows you to show and hide the content in-line rather than linking to another page. Collapsible elements can be represented as a stand-alone block or as a set of connected blocks.

To create a collapsible item, all you need to do is add `data-role="collapsible"` to a `<div>` element. The `<div>` element needs to have a header to display in a bar when it's collapsed. To group items, you add multiple `<div data-role="collapsible">` elements inside a `<div data-role="collapsible-set">` element.

To force an item to be expanded, add `data-collapsed="false"`. This sets the initial state to expanded, although you can still collapse and expand it by clicking on the header.

Also, you can control the themes used to render the collapsible sets using `data-theme` to define the header and `data-content-theme` to define the collapsed content.

Adding Auxiliary Content to Panels

```
<div data-role="panel" id="config"
    data-position="right" data-display="reveal">
  <div data-role="header" data-theme="a">
    <h3>Panel</h3></div>
  <h3>Settings</h3>
  <label for="Option1">Option 1</label>
  <input type="checkbox" id="Option1"></input>
  <label for="Option2">Option 2</label>
  <input type="checkbox" id="Option2"></input>
  <a data-role="button" data-icon="delete"
    data-rel="close" data>Close Config</a>
</div>
```

A useful way to present data that is not necessarily part of the page but is relevant is panels. A panel is similar to the page but sits off to the left or right side. When opened, the panel reveals the additional information.

Panels are defined using `data-role="page"` and must be siblings to the header, content, and footer elements inside a mobile page. Panels are opened by linking to the `id` value, similar to opening a new page. When the link is clicked, the panel is displayed using one of the following three display modes:

- **`data-display="overlay"`**—Panel elements overlay the existing page with a transparent background.
- **`data-display="push"`**—Panel content "pushes" the existing page as it is exposed.
- **`data-display="reveal"`**—Panel sits under the current page and is revealed as the current page slides away.

The panel is positioned using `data-position="right"` or `data-position="left"`. When opened, it scrolls with the page. You can force a fixed position using `data-position-fixed="true"`, in which case the panel contents appear relative to the screen and not the scroll position.

To close the panel, add a link button with the `<a data-rel="closed">` attribute set to the panel page. You can also close a panel from jQuery code using the following:

```
$( "#panelId" ).panel( "close" );
```

Working with Popups

```
<div data-role="panel" id="config"
    data-position="right" data-display="reveal">
  <div data-role="header" data-theme="a">
    <h3>Panel</h3></div>
  <h3>Settings</h3>
  <label for="Option1">Option 1</label>
  <input type="checkbox" id="Option1"></input>
  <label for="Option2">Option 2</label>
  <input type="checkbox" id="Option2"></input>
  <a data-role="button" data-icon="delete"
    data-rel="close" data>Close Config</a>
</div>
```

One of your best friends when implementing mobile page content is the popup. A popup is different from a panel in that it can be displayed anywhere on the page that's currently being viewed. This allows you to add additional bits of extra information that the user can easily click and see.

Popups are defined using `data-role="popup"` and can be placed anywhere inside the content of a mobile page. Popups are also opened by linking to the ID value of the `<div data-role="popup">` tag. However, you must add a `data-rel="popup"` to the `<a>` tag that links to the popup. You can also manually open a popup using the following from jQuery code:

```
$("#popupId").popup();
```

The popup is positioned using `data-position-to attribute` that can be set to window, origin, or the `#id` of an element. jQuery Mobile tries to center the popup over that element.

To close the popup, simply click on the page somewhere other than the popup. Also, you can add a Close button to the popup `<div>` by adding the

data-rel="back" attribute. For example, the following code adds a Close button and specifies the delete icon and notext:

```
<a href="#" data-rel="back" data-role="button" data-
➥theme="a"
   data-icon="delete" data-iconpos="notext"
➥class="ui-btn-right">Close</a>
```

The following shows an example creating a popup menu of links that tie to photo popups, as illustrated in Figure 14.6. One thing to note is that the image popup has a class="photopopup". This provides class settings to style the image popup container.

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script src="../js/jquery.mobile.js"></script>
10     <link rel="stylesheet"
11       href="../js/css/jquery.mobile.css" />
12   <script>
13     $(document).ready(function() {
14       checkForMobile();
15     });
16   </script>
17   <style>
18     .photopopup img { width:220px; }
19     #menu-popup {width:200px; }
```

```
20   </style>
21 </head>
22 <body>
23   <div id="border"><div id="frame">
24   <div data-role="page" id="pageOne">
25     <div data-role="header"><h1>Popups</h1></div>
26     <div data-role="content">
27       <a data-role="button" href="#menu" data-
➥rel="popup"
28         id="menuLink">Popup Menu</a>
29       <div data-role="popup" id="menu"
30         data-position-to="#menuLink">
31         <ul data-role="listview">
32           <li><a href="#photo1" data-rel="popup"
33             id="imageLink">Image 1</a></li>
34           <li><a href="#photo2" data-rel="popup"
35             id="imageLink">Image 2</a></li>
36         </ul></div>
37       <div data-role="popup" id="photo1"
38         data-position-to="window" class=
➥"photopopup">
39         <a href="#" data-rel="back" data-
➥role="button"
40           data-icon="delete" data-
➥iconpos="notext"
41           class="ui-btn-right">Close</a>
42         <img src="../images/falls.jpg" /></div>
43       <div data-role="popup" id="photo2"
44         data-position-to="window" class=
➥"photopopup">
45         <a href="#" data-rel="back" data-
➥role="button"
46           data-icon="delete" data-
➥iconpos="notext"
47           class="ui-btn-right">Close</a>
48         <img src="../images/peak.jpg" /></div>
49     </div>
```

```
50    </div>
51    </div></div>
52 </body>
53 </html>
```

ch1405.html

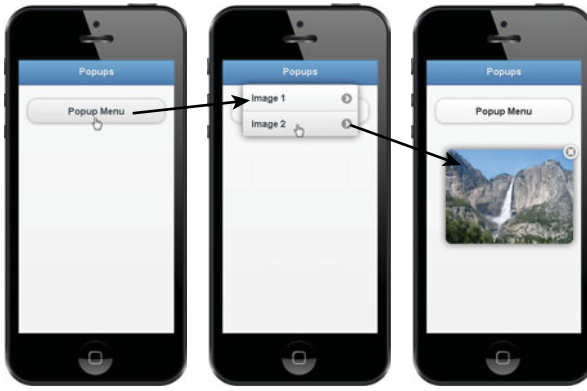


Figure 14.6 Creating a menu popup that links to photo popups using jQuery Mobile code in ch1405.html.

Building Mobile-Friendly Tables

```
<table data-role="table" data-mode="reflow"
    id="parkTable">
 . . . or . . .
<table data-role="table" data-mode="columntoggle"
    id="parkTable">
  <thead><tr><th>Park</th>
    <th data-priority="1">State</th>
    <th data-priority="2">Est.</th>
    <th data-priority="3">Photo</th></tr>
  </thead>
  <tbody>
```

```
    <tr><td>Yellowstone</th><td>MT</td><td>1872</td>
      <td><img src="bison.jpg" width="100" /></tr>
    <tr><td>Yosemite</th><td>CA</td><td>1872</td>
➥<td>
      <img src="falls.jpg" width="100" /></tr>
    <tr><td>Zion</th><td>UT</td><td>1919</td><td>
      <img src="peak.jpg" width="100" /></tr>
  </tbody>
</table>
```

jQuery Mobile adds a `data-mode` attribute to the `<table>` tag to allow for mobile-friendly tables. There are two types of ways to make a table mobile-friendly.

The first is to reduce thenumber of columns displayed, and the second is to collapse rows into column sets so they can be stacked on each other. This is done by adding the `data-role="table"` and `data-mode="reflow"` to the `<table>` tag. Cells in the table are repositioned so that they flow with the rest of the page. In reflow mode, when the table is 2 columns wide, the columns are broken up into individual and cells stacked on top of each other. Then the headers in the column are added as labels to the cells so that each cell contains the column header to the left to identify the value.

The second solution is to add a `data-mode="columntoggle"` to the `<table>` tag. This provides the user with a popup menu so the user can enable/disable columns to display. The columns that can be enabled/disabled in `columntoggle` mode are designated by adding a `data-priority=#` attribute to the `<th>` items in the first `<tr>` of the `<thead>` element. You cannot disable columns without the `data-priority`. The `#` value of `data-priority` ranges from `1` (highest) to `6` (lowest).

Creating Mobile Forms

jQuery Mobile introduces several attributes for elements that extend and define the behavior to support mobile devices. Table 14.4 lists some of those attributes that you need to be familiar with as you implement mobile forms. The list is not comprehensive.

Table 14.4 **jQuery Mobile Data Attributes for Form Elements**

Attribute	Description
`data-role="fieldcontain"`	Allows for multiple form elements to be styled as a single group.
`data-role="controlgroup"`	Allows you to group buttons into a single block similar to a navigation bar.
`data-type`	Specifies if the items in the `controlgroup` should be organized `"vertical"` or `"horizontal"`.
`data-corners="true"`	Adds a corner radius to elements.
`data-icon`	Specifies an icon to be added to an element: `home`, `delete`, `plus`, `arrow-u`, `arrow-d`, `check`, `gear`, `grid`, `star`, `custom`, `arrow-r`, `arrow-l`, `minus`, `refresh`, `forward`, `back`, `alert`, `info`, `search`
`data-iconpos`	Specifies the location the icon is placed: `left, right, top, bottom`

Attribute	Description
`data-mini`	Boolean. If `true`, a mini version of the element is rendered with limited padding and margins.
`data-theme`	Applies a theme swatch to the element (a–z).

Adding/Hiding Labels

```
<div data-role="fieldcontain" class="ui-hide-label">
  <label for="username">Username:</label>
  <input type="text" name="username" id="username"
    value="" placeholder="Username"/>
</div>
```

Labels are required on form inputs in jQuery Mobile. This allows the library to format the form elements appropriately for mobile devices. Therefore, you need to add a label and use the `for` attribute to link it to the form input. The good news is that jQuery mobile provides a class that you can easily hide the label with.

Disabling Form Elements

```
<input type="text" name="username" id="username"
➥value=""
    placeholder="Username" class="ui-disabled" />
```

You can disable form elements by adding the `ui-disabled` class to them in the HTML definition or programmatically in your jQuery code. Disabling the elements prevents the control from accepting input from the user. Also, jQuery Mobile has special styling that is applied to disabled controls to make it apparent to the user that the form element cannot be used.

Refreshing Form Elements

```
var mySelect = $("#mySelect");
mySelect [0].selectedIndex = 1;
mySelect.selectmenu("refresh");
$("#myCheckbox").prop("checked",true).checkboxradio
➥("refresh");
$("#myRadio").prop("checked",true).checkboxradio("re
➥fresh");
$("#mySlider").val(100).slider("refresh");
```

When programmatically changing items in a form, such as adding options to a select, you need to call the refresh function on the item. Following are some examples of adjusting form elements using jQuery and then refreshing them.

Adding Form Elements

jQuery Mobile elements are for the most part standard. jQuery Mobile formats them for better display on mobile devices. However, there are a few data attributes that you should be aware of:

- **`data-role="fieldcontain"`**—Add to a container class around multiple form elements. Helps mobile elements, especially `<label>` elements, track closer.
- **`class="ui-hide-label"`>**—Adding this class to a `fieldcontain` container causes the label to hide.
- **`data-role="button"`**—Forces `<a>` links to be displayed as mobile buttons.
- **`data-icon`**—Specifies an icon that can be added to a button or link.
- **`data-iconpos`**—Specifies the location of the icon.

- **`data-role="controlgroup"`**—Add to a container class around multiple form elements. The elements are rendered as a single set. You can also add `data-type="horizontal"` and `data-type="vertical` to define whether the buttons are stacked on top of each other or in a row.

The best way to illustrate a mobile form is to show you an example. The following code implements several of the HTML form elements and applies the jQuery Mobile attributes to them. The rendered version of the form is shown in Figure 14.7:

```
01 <html>
02 <head>
03   <title>Python Phrasebook</title>
04   <meta charset="utf-8" />
05   <meta name="viewport" content="width=device-
➥width,
06     initial-scale=1">
07   <script src="../js/jquery-
➥2.0.3.min.js"></script>
08   <script src="../js/initmobile.js"></script>
09   <script src="../js/jquery.mobile.js"></script>
10     <link rel="stylesheet"
11       href="../js/css/jquery.mobile.css" />
12   <script>
13     $(document).ready(function() {
14       checkForMobile();
15     });
16   </script>
17   <style>
18   </style>
19 </head>
20 <body>
21 <div id="border"><div id="frame">
```

```
<div data-role="page" id="pageOne">
  <div data-role="header"><h1>Form</h1></div>
  <div data-role="content">
    <div data-role="controlgroup"
        data-type="horizontal" data-mini="true">
      <button data-icon="grid"
        data-iconpos="notext">b</button>
      <button data-icon="gear"
        data-iconpos="notext">b</button>
      <button data-icon="star"
        data-iconpos="notext">b</button>
    </div>
    <div data-role="fieldcontain" class="ui-hide-
➥label">
        <label for="search">Search</label>
        <input type="search" name="search"
          id="search" value="" />
    </div>
    <div data-role="fieldcontain" class="ui-hide-
➥label">
      <input type="text" name="username"
➥id="username"
        placeholder="your name here"/>
      <label for="username">Name</label>
    </div>
    <fieldset data-role="controlgroup" data-
➥mini="true">
     <input type="radio" name="rc" id="rc1"
➥value="rc1"/>
      <label for="rc1">Family</label>
     <input type="radio" name="rc" id="rc2"
➥value="rc2"/>
      <label for="rc2">Friend</label>
    </fieldset>
    <div data-role="controlgroup" data-type=
➥"horizontal"
        data-mini="true" data=theme="a">
```

```
52       <input type="checkbox" name="hcb" id="hcb1"
➥/>
53       <label for="hcb1">Post</label>
54       <input type="checkbox" name="hcb" id="hcb2"
➥/>
55       <label for="hcb2">Link</label>
56       <input type="checkbox" name="hcb" id="hcb3"
➥/>
57       <label for="hcb3">Share</label>
58     </div>
59     <div data-role="fieldcontain" class="ui-hide-
➥label">
60       <label for="area">Comments:</label>
61       <textarea name="textarea" id="area" data-
➥theme="a">
62         Comments</textarea>
63     </div>
64     <button data-inline="true"
65       data-theme="b">Accept</button>
66     <button data-inline="true"
67       data-theme="e">Decline</button>
68 </div>
69 </div></div>
70 </body>
71 </html>
```

ch1406.html

Figure 14.7 Creating basic mobile form elements using the jQuery Mobile code in ch1406.html.

Index

A

absolute values, calculating, 34

accessing

- browsers, 47-49
  - accessing cookies, 52-55
  - history, 49
- HTML
  - chaining object operations, 75-76
  - elements, 59
  - navigating objects, 76-82
  - searching elements, 59-68, 71-74
- jQuery in JavaScript, 6
- libraries, 7
- Mobile, 10
- variables, 16

addEventListener() function, 103

adding

- auxiliary content to panels, 328
- borders, 134
- conditional blocks of code, 30
- cookies, 53
- dividers to lists, 324
- DOM elements to objects, 91
- elements to elements, 179-183
- event handlers, 99, 105
- forms
  - elements, 336
  - mobile, 334-336, 340
- initialization code, 100
- items to lists, 191-193
- JavaScript
  - to HTML documents, 3
  - loading from external files, 4
- jQuery to web pages, 5-6
- labels, 335
- mouse-click-handling code, 115
- navigation buttons, 307, 311, 314
- page load event handlers, 99
- rows to tables, 189-191
- tables, 333
- timers, 55-57
- transitions, 308

UI elements
applying sliders, 215-216, 219
attaching datepicker, 212-215
coding tooltips, 223-225
creating menus, 220-222
downloading libraries, 201-202
dragging/dropping, 205-212
implementing auto-complete, 203-205
jQuery, 201

adjusting opacity, 146-149. *See also* modifying

adjValues() function, 197

AJAX
asynchronous communication, 253-254
cross-domain requests, 254
GET/POST requests, 255
JavaScript, 261-267
jQuery, 267-289
overview of, 251-252
response data types, 256-259

ancestors, searching, 80

.animate() method, 228-229

animation, 227-228
.hide() method, 242
.show() method, 243
.toggle() method, 243-245
CSS settings, 228-229
delaying, 233
images
moving elements, 248-250
resizing, 246-248
queues, 231
sliding toggles, 239-242
stopping, 232-233
visibility, 234-238

appending
bottom of element's content, 180
elements, 141
rows to tables, 189-191
text, 140

applications, Mobile, 9-12

applying
AJAX, 251-252
asynchronous communication, 253-254
cross-domain requests, 254
GET/POST requests, 255
JavaScript, 261-267
jQuery, 267-289
response data types, 256-259
event objects, 111-114
filters to selectors, 74
for() loops, 31
grid layouts, 316-317, 320

logic, 29-31
.map() method, 87-88
nested lists, 322
popups, 329, 332
power functions, 35
right-click, 117-118
selectors, 62
sliders, 215-216, 219
trigonometric functions, 35
while() loops, 30
arguments, 20
arrays
combining, 26
creating, 18
items
deleting, 27
detecting, 27
iterating, 31
manipulating, 25-27
sorting, 28
splicing, 26
assigning
data values to objects, 89
event handlers in HTML, 101
asynchronous communication, 253-254
attaching datepicker element, 212-215
.attr() method, 126, 166
attributes, selecting based on HTML, 64
autocomplete, implementing, 203-205

B

back buttons, creating, 309
behavior, events, 107-110
blocks, adding code, 30
.blur() method, 173
borders, adding, 134
bottom of element's content, appending, 180
browsers. *See also* interfaces
accessing, 47-49
cookies, modifying, 52-55
current location details, 45-47
development tools, configuring, 12-13
events
adding event handlers, 99-106
forms, 122-123
keyboards, 118-120
managing, 107-110
mouse, 115-118
objects, 111-114
overview of, 96
types, 96, 99
history, accessing, 49
JavaScript consoles, applying, 44
navigating, 43
popup windows, creating, 50-51
screens, sizing, 45
timers, adding, 55, 57

web pages
redirecting, 44
reloading, 44
building mobile pages, 302-304
buttonImage option, 213
buttonImageOnly option, 213
buttons
back, creating, 309
navigation
adding, 307, 311, 314
positioning, 308
split lists, 324

C

calculating absolute values, 34
calling functions, 20
cancelling timers, 56
Cascading Style Sheets. *See* CSSs
case, modifying strings, 24
CDNs (Content Discovery Networks), 5, 9
chaining object operations, 75-76
change option, 216
.changePage() method, 298
changes to text, detecting, 119
characters
searching, 22
special, string objects, 21
check box state, modifying, 161-162
checking
items in arrays, 27
for substrings, 25
children, retrieving elements, 77
Chrome, enabling JavaScript in, 13
classes
deleting, 137
names, searching DOM objects, 60
toggling, 137, 139
click() method, 110
closest elements, retrieving, 77
closing windows, 49
code
blocks, adding, 30
dynamic programming. *See* dynamic programming
initialization, adding, 100
JavaScript
adding to HTML documents, 3
consoles, 44
loading from external files, 4
overview of, 2
mouse-click-handling, adding, 115
collapsible elements, dividing content into, 326-327
colors, modifying, 131-132

combining
arrays, 26
strings, 23
communication, asynchronous, 253-254
complete function, 229-230
components, dates, 39
conditional blocks of code, adding, 30
configuring
browser development tools, 12-13
cookie values, 53
CSS properties, 130-139
default mobile settings, 301
DOM element properties, 126-129
hidden form attributes, 166
select inputs, 164-165
selected option in radio groups, 162
text
input values, 160
status bars, 48
timers, 55-57
consoles, JavaScript, 44
content
auxiliary, adding to panels, 328
HTML elements, selecting based on, 66
parent, appending elements, 141
web pages
adding elements, 179-183
appending rows to tables, 189-191
building dynamically, 177
deleting elements, 184-185
HTML elements, 178
HTML5 canvas graphics, 197-199
image galleries, 193-196
inserting items into lists, 191-193
select form elements, 186-188
Content Discovery Networks. *See* CDNs
content tooltip, 223
converting
DOM objects into jQuery objects, 84
numbers to strings, 22
strings to numbers, 23
coordinates, getting mouse, 115
cross-domain requests, AJAX, 254
.css() method, 130
CSSs (Cascading Style Sheets), 7
animating, 228-229
elements, 130-139
theme swatches, 295

current date and time, getting, 37
current hashes, searching, 45
current location details, 45-47
current location of web pages, 48
customizing. *See also* configuring
forms, 334-336, 340
popups, 329-332
tables, 333

D

.data() method, 166
data types, AJAX, 256-259
Date object, 36-40
dateFormat option, 213
datepicker element, attaching, 212-215
dates, components, 39
dblclick() method, 110
debugging JavaScript consoles, 44
default behavior, stopping, 109
default mobile settings, configuring, 301
defining
functions, 20
variables, 16
viewport meta tags, 300
delay timers, adding, 56
delaying animation, 233
deleting
classes, 137
elements to elements, 184-185
event handlers
JavaScript, 104
jQuery, 106
items from arrays, 27
objects, 91
deltas, formatting time, 39
descendent elements, searching, 78
detecting
changes to text, 119
items in arrays, 27
mobile screen size, 294
Developer Tools (Internet Explorer), 13
development
browser tools, configuring, 12-13
jQuery Mobile, 9-12
devices, Mobile, 9-12
disabling form elements, 167-169, 335
dividers, adding to lists, 324
dividing content into collapsible elements, 326-327
Document Object Model. *See* DOM
documents, adding JavaScript to HTML, 3

DOM (Document Object Model), 5
elements
adding, 91
configuring properties, 126-129
content, 139-141
objects, 60-61, 84
downloading jQuery UI libraries, 201-202
dragging/dropping elements, 205-212
droppable widget options, 208
duration function, 229-230
dynamic programming
forms
check box state, 161-162
disabling elements, 167-169
elements, 159
forcing focus to/away from elements, 172-174
hidden attributes, 166
managing submissions, 175
radio inputs, 162
select values, 164-165
showing/hiding elements, 170-172
text input values, 160
web pages, 177
adding elements, 179-183
adjusting opacity, 146-149
appending rows to tables, 189-191
CSS properties, 130-139
deleting elements, 184-185
DOM element properties, 126-129
element content, 139-141
hiding/viewing elements, 144-146
HTML elements, 178
HTML5 canvas graphics, 197-199
image galleries, 193-196
inserting items into lists, 191-193
modifying, 125
modifying layouts, 143
repositioning elements, 152-153, 156
resizing elements, 149, 152
select form elements, 186-188
stacking elements, 156-158

E

.each() method, 85
easing function, 229-230
effects, animation, 227-228
CSS settings, 228-229
delaying, 233
.hide() method, 242
moving elements, 248-250
queues, 231
resizing images, 246-248
.show() method, 243
sliding toggles, 239-242
stopping, 232-233
.toggle() method, 243-245
visibility, 234-238
elements
appending, 141
children, retrieving, 77
content, 139-141
CSS properties, 130-139
descendent, searching, 78
DOM, 5
adding, 91
configuring properties, 126-129
fading, 239, 242
forms, 159
adding, 336
check box state, 161-162
creating select, 186-188
disabling, 167-169, 335
forcing focus to/away from elements, 172-174
hidden attributes, 166
managing submissions, 175
radio inputs, 162
refreshing, 336
select values, 164-165
showing/hiding, 170-172
text input values, 160
HTML
adding, 179-183
deleting, 184-185
jQuery, 178
moving, animating, 248-250
UIs
adding, 201
applying sliders, 215-219
attaching datepicker, 212-215
coding tooltips, 223-225
creating menus, 220-222
downloading libraries, 201-202
dragging/dropping, 205-212
implementing autocomplete, 203-205

web pages
hiding/viewing, 144-146
loading HTML into, 269-271
repositioning, 152-156
resizing, 149-152
stacking, 156-158
enabling
Developer Tools on Internet Explorer, 13
JavaScript in Chrome, 13
.eq (index) filter, 92
equality, objects, 29
event handlers
global, 285
swipe, 310, 314
event.preventDefault() method, 109
event.stopPropagation() method, 109
events
browsers
adding event handlers, 99-106
forms, 122-123
keyboards, 118-120
managing, 107-110
mouse, 115-118
objects, 111-114
overview of, 96
types, 96, 99
draggable widget, 207
droppable widget, 209
mobile, 295-297
reset, 175
submit, 175
existingObject, 180
external files, loading from JavaScript, 4

F

fading
animation elements in/out, 234-235
elements, 239-242
to levels of opacity, 236
files
accessing, 6
JavaScript, loading from external, 4
paths, viewing, 46
web pages, loading, 5-6
.filter(filter) method, 92
filters
object results, 92-94
selectors, applying, 74
finding. *See* searching
Firefox, installing Firebug, 13
.first() method, 93
focus, modifying, 122
.focus() method, 173
fonts, modifying, 136
footers, mobile web pages, 304
for() loops, 31
forcing focus to/away from form elements, 172-174

formatting
arrays, 18
back buttons, 309
dates, 36-40
HTML
adding elements, 179-183
deleting elements, 184-185
elements, 178
image galleries, 193-196
listviews, 320-325
menus, 220-222
mobile pages, 302-304
navbars, 314
objects, 19
popup windows, 50-51
strings, modifying, 21-25
text, input values, 160
time
deltas, 39
strings, 38
tooltips, 223-225
forms
elements, 159
adding, 336
check box state, 161-162
creating select, 186-188
disabling, 167-169, 335
forcing focus to/away from elements, 172-174
hidden attributes, 166
managing submissions, 175
radio inputs, 162
refreshing, 336
select values, 164-165
showing/hiding, 170-172
text input values, 160
events, 122-123
HTML elements, selecting based on, 71
mobile, adding, 334-336, 340
functionality, AJAX, 285-289
functions
addEventListener(), 103
adjValues(), 197
calling, 20
complete, 229-230
defining, 20
duration, 229-230
easing, 229-230
handler, 105
power, applying, 35
queue, 230
removeEventListener() function, 104
renderSpark(), 197
trigonometric, applying, 35

G

galleries, formatting image, 193-196
generating random numbers, 32
GET requests, AJAX, 255, 262-264
.get() method, 84
JSON, handling, 271-274
XML, handling, 274, 277
.getScript() method, 256
getting
CSS properties, 130-139
DOM element properties, 126-129
Elements, content, 139-141
event target objects, 114
hidden form attributes, 166
mouse coordinates, 115
select inputs, 164-165
selected option in radio groups, 162
text input values, 160
global event handlers, 285
global setup, modifying, 285
graphics, HTML5 canvas, 197-199
grids, layouts, 316-320
groups, getting/setting radio options, 162

H

handlers
functions, 105
events
adding, 99
swipe, 310-314
global event, 285
handling
events
forms, 122-123
keyboards, 118-120
mouse, 115-118
JSON data, 271-274
selection changes, 123
text data (AJAX), 257
XML data, 274-277
.has(selector or element) method, 93
hashes, current, 45
headers, mobile web pages, 304
hidden form attributes, 166
.hide() method, 144, 170, 242
hiding
elements, web pages, 144-146
form elements, 170-172
labels, 335
hierarchies, selecting based on HTML elements, 68
history, navigating browsers, 49

hosts, searching names, 46
HTML (Hypertext Markup Language)
elements
accessing, 59
adding, 179-183
appending/prepending text, 140
chaining object operations, 75-76
deleting, 184-185
getting content of, 140
jQuery, 178
navigating objects, 76-82
repositioning, 152-156
resizing, 149, 152
searching, 59-68, 71-74
stacking, 156-158
event handlers, assigning in, 101
JavaScript, adding to, 3
response data, 259
HTML5 canvas graphics, 196-199
.html() method, 140

I

ID, searching DOM objects, 60
images
galleries, creating, 193-196
resizing, animating, 246-248
source files, modifying, 128
transitions, adding, 237-238
implementing
autocomplete, 203-205
low-level AJAX requests, 287-289
mobile sites with multiple pages, 306-314
searchable lists, 325
split button lists, 324
initialization code, adding, 100
innerHeight, 47
innerWidth, 47
input
autocomplete, implementing, 203-205
radio, 162
select, 164-165
text values, 160
inserting into middle of element's content, 181
installing Firebug on Firefox, 13
interfaces
browser development tools, configuring, 12-13
loading, 9
navigating, 7
Internet Explorer, enabling Developer Tools, 13

.is() method, 161-163
items
arrays
deleting, 27
detecting, 27
lists, 191-193
tooltip, 223
iterating
arrays, 31
jQuery objects, 85-86
through object properties, 31

J

JavaScript
AJAX from, 261-267
arrays
creating, 18
manipulating, 25-27
Chrome, enabling in, 13
consoles, 44
Date object, 36-40
event handlers
adding, 103
deleting, 104
events, 96, 99
adding page load event handlers, 99
managing, 107-110
external files, loading from, 4
functions, defining, 20
GET requests, sending from, 262-264
HTML
adding to documents, 3
searching elements, 59-61
jQuery, accessing, 6
logic, applying, 29-31
math operations, 31-35
objects, creating, 19
on-demand, 255
overview of, 2
POST requests, sending from, 264, 267
strings, manipulating, 21-25
syntax, 15
variables, defining, 16
JavaScript Object Notation. *See* JSON
.join() method, 27
jQuery
AJAX from, 267-289
animation, 227-228
CSS settings, 228-229
delaying, 233
.hide() method, 242
queues, 231
.show() method, 243
sliding toggles, 239-242
stopping, 232-233
.toggle() method, 243-245
visibility, 234-238

event handlers
adding, 105
deleting, 106
events, 96, 99
HTML elements, 178
initialization code, 100
JavaScript, accessing, 6
Mobile, 9-12
mobile web sites, 291
applying grid layouts, 316-320
building web pages, 302-304
creating navbars, 314
customizing popups, 329, 332
dividing into collapsible elements, 326-327
formatting listviews, 320-325
forms, 334-340
implementing with multiple pages, 306-314
overview of, 291-300
tables, 333
viewing panels, 328
objects
adding DOM elements to, 91
applying.map() method, 87-88
assigning data values to, 89
chaining operations, 75-76
converting DOM objects into, 84
deleting, 91
filtering results, 92-94
getting, 84
iterating, 85-86
modifying, 83
navigating to select elements, 76-82
overview of, 4
UIs
accessing libraries, 7
adding, 201
applying sliders, 215-219
attaching datepicker, 212-215
coding tooltips, 223-225
creating menus, 220-222
downloading libraries, 201-202
dragging/dropping, 205-212
implementing autocomplete, 203-205
loading, 9
navigating, 7
web pages, loading, 5-6
jQuery Selector, searching HTML elements, 61-74
JSON (JavaScript Object Notation)
handling, 271-274
response data, 257-258

JSON.parse() method, 258
JSONP (JSON with Padding), 255

K–L

keyboard events, 118-120
keys, pressing, 120
keywords, 17, 96, 99

labels, adding/hiding, 335
languages
JavaScript
adding to HTML documents, 3
loading from external files, 4
overview of, 2
syntax, 15
.last() method, 93
layouts
grids, applying, 316-317, 320
web pages
adjusting opacity, 146-149
hiding/viewing elements, 144-146
modifying, 143
repositioning elements, 152-153, 156
resizing elements, 149-152
stacking elements, 156-158
length, strings, 22
levels, tolerance, 208
libraries
accessing, 7
CDNs, loading, 5
jQuery Mobile, 291
applying grid layouts, 316-320
building web pages, 302-304
creating navbars, 314
customizing popups, 329-332
dividing into collapsible elements, 326-327
formatting listviews, 320-325
forms, 334-340
implementing with multiple pages, 306-314
overview of, 291-300
tables, 333
viewing panels, 328
jQuery UI, adding, 201-202
loading, 9
Mobile, 9-12
links, modifying locations, 127
lists
dividers, adding, 324
items, inserting into, 191-193
nesting, 322

searchable, implementing, 325
split button, 324
listviews, formatting, 320-325
.load() method, 101
loading
HTML into page elements, 269-271
JavaScript from external files, 4
jQuery in web pages, 5-6
libraries, 5, 9
Mobile, 12
mobile pages without displaying, 299
.loadPage() method, 298
location.reload() method, 44
locations
current location, 45-47
links, modifying, 127
web pages, 47-48
logic, applying, 29-31
loops
for(), 31
while(), 30
low-level AJAX requests, 287-289

M

managing
events, 107-110
form submissions, 175
manipulating
arrays, 25-27
cookies, 52-55
strings, 21-25
.map() method, applying, 87-88
Math object, 31-35
max option, 216
maximum numbers in sets, 34
menus, formatting, 220-222
meta tags, defining viewports, 300
methods
.animate(), 228-229
.attr(), 126, 166
autocomplete, 203
.blur(), 173
.changePage(), 298
click(), 110
.css(), 130
.data(), 166
.datepicker(), 213
dblclick(), 110
.delay(), 234
.each(), 85
event.preventDefault(), 109
event.stopPropagation(), 109
.filter(filter), 92
.first(), 93
.focus(), 173

.get()
JSON data, 271-274
XML data, 274, 277
.get([index]), 84
.getScript(), 256
.has(selector or element), 93
.hide(), 144, 170, 242
.html(), 140
.is(), 161-163
.join(), 27
JSON.parse(), 258
.last(), 93
.load(), 101
.loadPage(), 299
location.reload(), 44
.map(), 87-88
.not(filter) method, 94
.off(), 106
.offset(), 153
.on(), 105
onloadHandler(), 100
pop(), 27
.position(), 153
.post(), 278-281
.prop(), 126, 166
push(), 18
.ready(), 101
.remove(), 184
removeAttr(), 162
setTimeout(), 197
.show(), 144, 170, 243
sort(), 28
.slice(start, [end]), 94
.slideToggle(), 239
.toggle(), 243, 245
.toLowerCase(), 24
.toUpperCase(), 24
.val(), 160, 164

middle of element's content, inserting into, 181

minimum numbers in sets, 33

Mobile (jQuery), 9-12

mobile web sites (jQuery), 291
overview of, 291-300
web pages
applying grid layouts, 316-320
building, 302-304
creating navbars, 314
customizing popups, 329-332
dividing into collapsible elements, 326-327
formatting listviews, 320-325
forms, 334-340
implementing with multiple, 306-314
tables, 333
viewing panels, 328

modifying
check box state, 161-162
colors, 131-132
cookies, 52-55

elements
applying sliders, 215-219
content, 139-141
focus, 122
fonts, 136
global setup, 285
image source files, 128
link locations, 127
mobile web pages, 298
objects, 83
adding DOM elements to, 91
applying.map() method, 87-88
assigning data values to, 89
converting DOM objects into, 84
deleting, 91
filtering results, 92-94
getting, 84
iterating, 85-86
radio inputs, 162
select inputs, 164-165
selections, 123
strings, 21-25
text, detecting, 119
web pages
adjusting opacity, 146-149
CSS properties, 130-139
DOM element properties, 126-129
dynamic programming, 125
element content, 139-141
hiding/viewing elements, 144, 146
layouts, 143
repositioning elements, 152-153, 156
resizing elements, 149-152
stacking elements, 156-158

mouse
coordinates, getting, 115
events, 115-118
mouse-click-handling code, adding, 115
mouseout events, 116
mouseover events, 116

moving
elements, animating, 248-250
HTML elements, 152-156

N

names
attributes, 166
classes, searching DOM objects, 60
hosts, searching, 46
tags, searching DOM objects, 61

navbars, formatting, 314

navigating
browsers, 43
accessing, 47-49
adding timers, 55-57
applying JavaScript consoles, 44
current location details, 45-47
history, 49
modifying cookies, 52-55
popup windows, 50-51
redirecting web pages, 44
reloading web pages, 44
sizing screens, 45
jQuery Mobile library, 291-300
objects to select elements, 76-82
UIs, 7
accessing libraries, 7
loading, 9
navigation buttons
adding, 307-314
positioning, 308
nested lists, applying, 322
networks, CDNs, 5
.not(filter) method, 94
numberOfMonths option, 213
numbers
dates, formatting, 36-40
maximum in sets, 34
minimum in sets, 33
random, generating, 32
rounding, 33
strings, converting, 22

O

objects
creating, 19
Date, 36-40
DOM, 60-61
elements, navigating to select, 76-82
equality, 29
events, 111-114
existing, 180
getting, 84
jQuery
adding DOM elements to, 91
applying map() method, 87-88
assigning data values to, 89
converting DOM objects into, 84
deleting, 91
filtering results, 92-94
iterating, 85-86
location, 45-47
Math, 31, 34-35
modifying, 83
operations, chaining, 75-76
properties, iterating, 31

strings, special characters, 21
window.XMLHttpRequest, 261
windows, 47-49
XMLHttpRequest, 261
.off() method, 106
.offset() method, 153
.on() method, 105
on-demand JavaScript, 255
onloadHandler() method, 100
onSelect option, 213
opacity
adjusting, 146-149
fading to levels of, 236
opening windows, 49
operations
math, 31-35
objects, chaining, 75-76
options
animation, 229
draggable widget, 205
droppable widget, 208
radio groups, getting/setting, 162
orientation options, 216

P

pages
elements, loading HTML into, 269-271
load event handlers, adding, 99
panels, viewing, 328
parents
content, appending elements, 141
searching, 80
paths, viewing files, 46
physical events, 107. *See also* events
pop() method, 27
popups
creating, 50-51
applying, 329, 332
position tooltip, 223
.position() method, 153
positioning
HTML elements, 152-156
navigation buttons, 308
parents
retrieving, 80
searching, 80
POST requests, AJAX, 255, 264, 267
.post() method, 278-281
power functions, applying, 35
prepending
content, 179
text, 140
pressing keys, 120
previous siblings, 81
programming
dynamic, 125. *See also* dynamic programming
JavaScript
adding to HTML documents, 3

loading from external files, 4
overview of, 2
syntax, 15
.prop() method, 126, 166
propagation, stopping event, 109
properties
CSS, getting/setting, 130-139
DOM, configuring elements, 126-129
objects, iterating, 31
push() method, 18

Q–R

queries, retrieving strings, 46
queues
animating, 231
functions, 230

radio inputs, 162
random numbers, generating, 32
range option, 216
.ready() method, 101
recurring timers, adding, 57
redirecting web pages, 44
refreshing form elements, 336
reloading web pages, 44
.remove() method, 184
removeAttr() method, 162
removeEventListener() function, 104
renderSpark() function, 197
replacing
strings, 25
text, 141
repositioning HTML elements, 152-156
requests, AJAX
cross-domain, 254
GET/POST, 255
JavaScript, 261-267
jQuery, 267-289
response data types, 256-259
reset event, 175
resizing
HTML elements, 149-152
images, animating, 246-248
responses, AJAX
jQuery, 282-284
data types, 256-259
results, filtering objects, 92-94
retrieving, 79
children elements, 77
cookie values, 53
parents, positioning, 80
previous siblings, 81
query strings, 46
siblings, 79-82
right-click, applying, 117-118
rounding numbers, 33
rows, appending tables, 189-191

S

screens
mobile, detecting size, 294
sizing, 45
searchable lists, implementing, 325
searching
ancestors, 80
characters, 22
current hashes, 45
current location of web pages, 48
descendent elements, 78
host names, 46
HTML
chaining object operations, 75-76
elements, 59-68, 71-74
parents, 80
strings, 25
secure locations, viewing from, 47
select form elements, creating, 186-188
select inputs, 164-165
selecting
elements, navigating objects to, 76-82
modifying, 123
selectors
applying, 62
filters, 74
sending
GET requests from JavaScript, 262-264
POST requests from JavaScript, 264-267
servers
AJAX
asynchronous communication, 253-254
cross-domain requests, 254
GET/POST requests, 255
JavaScript, 261-267
jQuery, 267-289
overview of, 251-252
response data types, 256-259
updating, 278-281
sets
maximum numbers in, 34
minimum numbers in, 33
setTimeout() method, 197
settings. *See also* configuring
CSS
animating, 228-229
properties, 130-139
DOM element properties, 126-129
hidden form attributes, 166
select inputs, 164-165
selected option in radio groups, 162
text input values, 160

setup, modifying global, 285
.show() method, 144, 170, 243
showButtonPanel option, 213
showOn option, 213
siblings
previous, 81
retrieving, 79-82
sizing
screens, 45, 294
web pages, 47
.slice(start, [end]) method, 94
slide option, 216
.slideToggle() method, 239
sliders, applying, 215-216, 219
sliding toggles, 239-242
sorting arrays, 28
source files, modifying images, 128
special characters, 21
special effects, animation, 227-228
CSS settings, 228-229
delaying, 233
.hide() method, 242
moving elements, 248-250
queues, 231
resizing images, 246-248
.show() method, 243
sliding toggles, 239-242
stopping, 232-233
.toggle() method, 243-245
visibility, 234-238
splicing
arrays, 26
strings, 24
split button lists, 324
splitting strings, 24
stacking HTML elements, 156-158
state, modifying check boxes, 161-162
status bars, configuring text, 48
step function, 231
stopping
animation, 232-233
default behavior, 109
event propagation, 109
strings
case modifying, 24
combining, 23
dates, formatting, 37
manipulating, 21-25
numbers, converting, 22
queries, retrieving, 46
replacing, 25
searching, 25
splicing, 24
splitting, 24
substrings, checking for, 25
time, formatting, 38
submit event, 175
submitting forms, managing, 175

substrings, checking for, 25
swatches, theme, 295
swipe event handlers, 310, 314
syntax, JavaScript, 15

T

tab-separated strings, creating arrays from, 27
tables
adding, 333
rows, appending, 189-191
tags, searching DOM objects, 61
targets, getting event objects, 114
text
AJAX, 257
appending/prepending, 140
autocomplete, implementing, 203-205
input values, 160
modifying, detecting, 119
replacing, 141
status bars, configuring, 48
theme swatches, 295
time
current, getting, 37
deltas, formatting, 39
strings, formatting, 38
timers, adding, 55-57
.toggle() method, 243-245
toggling
classes, 137-139
element visibility on/off, 235
tolerance levels, 208
.toLowerCase() method, 24
tools
browser development, configuring, 12-13
JavaScript consoles, 44
tooltips, formatting, 223-225
.toUpperCase() method, 24
transitions
adding, 308
animation, 237-238
triggering
animation, 232
events manually, 110
trigonometric functions, applying, 35
types of events, 96, 99

U

UIs (user interfaces)
elements
adding, 201
applying sliders, 215-219
attaching datepicker, 212-215
coding tooltips, 223-225
creating menus, 220-222

downloading libraries, 201-202
dragging/dropping, 205-212
implementing auto-complete, 203-205
libraries, accessing, 7
loading, 9
updating servers, 278-281
user interaction, adding, 179-183
user interfaces. *See* UIs

V

.val() method, 160, 164
value option, 216
values
absolute, calculating, 34
attributes, 166
cookies, configuring, 53
objects, assigning, 89
text input, 160
var keyword, 17
variables
accessing, 16
defining, 16
viewing
elements, web pages, 144-146
file paths, 46
form elements, 170-172
listviews, formatting, 320-325
mobile pages, loading without displaying, 299
panels, 328
web pages, 47
web sites on mobile devices, 294
viewports, defining meta tags, 300
visibility
animation, 234-238
HTML elements, selecting based on, 72

W

web forms, 159. *See also* forms
web pages. *See also* HTML
content
adding elements, 179-183
appending rows to tables, 189-191
building dynamically, 177
deleting elements, 184-185
HTML elements, 178
HTML5 canvas graphics, 197-199
image galleries, 193-196
inserting items into lists, 191-193
select elements, 186-188

current location of, 48
dynamic programming
CSS properties, 130-139
DOM element properties, 126-129
element content, 139-141
modifying, 125
events
adding event handlers, 99-106
forms, 122-123
keyboards, 118-120
managing, 107-110
mouse, 115-118
objects, 111-114
overview of, 96
types, 96-99
history, navigating, 49
HTML, loading into, 269-271
jQuery, loading, 5-6
layouts
adjusting opacity, 146-149
hiding/viewing elements, 144-146
modifying, 143
repositioning elements, 152-156
resizing elements, 149-152
stacking elements, 156-158
mobile, building, 302-304
navbars, formatting, 314
redirecting, 44
reloading, 44
secured locations, viewing from, 47
UI elements
adding, 201
applying sliders, 215-219
attaching datepicker, 212-215
coding tooltips, 223-225
creating menus, 220-222
downloading libraries, 201-202
dragging/dropping, 205-212
implementing autocomplete, 203-205
viewing, 47
web servers, AJAX
asynchronous communication, 253-254
cross-domain requests, 254
GET/POST requests, 255
JavaScript, 261-267
jQuery, 267-289
overview of, 251-252
response data types, 256-259

web sites, mobile (jQuery), 291. *See also* web pages
applying grid layouts, 316-320
building web pages, 302-304
creating navbars, 314
customizing popups, 329-332
dividing into collapsible elements, 326-327
formatting listviews, 320-325
forms, 334-340
implementing with multiple pages, 306-314
overview of, 291-300
tables, 333
viewing panels, 328
while() loops, 30
widgets
datepicker, attaching, 212-215
draggable/droppable, 205-212
menus, creating, 220-222
tooltips, creating, 223-225
window.XMLHttpRequest object, 261
windows
closing, 49
objects, 47-49
opening, 49
popup, creating, 50-51

X–Z

XML (Extensible Markup Language)
handling, 274, 277
response data, 259
XMLHttpRequest object, 261

z-index, 157