Adriaan de Jonge
Phil Dutson

Covers jQuery 1.6.4 and 1.7, with Tips for 1.8

jQuery, jQuery UI, and jQuery Mobile

Recipes and Examples

Developer's Library

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *jQuery, jQuery UI, and jQuery Mobile*

"de Jonge and Dutson's cookbook approach to jQuery, jQuery UI, and jQuery Mobile teaches how to create sophisticated projects quickly, removing all the guesswork. They really make jQuery seem effortless!"

—Joseph Annuzzi, Jr., Web architect, PeerDynamic.com

"This book is great for all aspects of jQuery development; it has it all, from a great UI section down to the most current tech, which is mobile. I found myself referencing this book numerous times for projects I am currently working on.

"This book will be excellent for anyone who is eager to learn more about jQuery and what the capabilities are. The authors put the learning in terms that anyone can understand and build from."

—Erik Levanger, UI UX engineer

"I have often said that when it comes to jQuery, half the battle is just being familiar with the types of things it is capable of doing. Having a good foundation can easily help you bridge the gap between 'This is going to be complicated' and 'I can do that with jQuery!' Like any good recipe book, this book includes ready-to-use code samples that demonstrate basic to advanced techniques. In addition to the major areas noted in the book title, there are references on how to customize the jQuery UI features to meet your needs and bonus sections about creating and using plugins. I also really appreciate the authors' insights regarding performance issues and best practices. Lastly, speaking as someone with no previous jQuery Mobile experience, this book provides solid examples to get you up and running fast. Overall, this book will help to greatly expand the skills in your jQuery arsenal."

—Ryan Minnick, software developer, Verizon Wireless

"The ultimate cookbook for anyone using jQuery, jQuery UI, or jQuery Mobile."

—Stretch Nate, Web applications developer, HealthPlan Services

"Through easy-to-understand recipes, de Jonge and Dutson give the reader a practical introduction to all things jQuery, from the most basic selectors to advanced topics, such as plugin authoring and jQuery Mobile. A great starting point for anyone interested in one of the most powerful JavaScript libraries."

—Jacob Seidelin, Web developer, Nihilogic

This page intentionally left blank

jQuery, jQuery UI, and jQuery Mobile

Developer's Library Series

Stephen G. Kochan
Second Edition
Programming in Objective-C 2.0
A complete introduction to the Objective-C language for Mac OS X and iPhone development
Developer's Library
Shane Conder
Lauren Darcey
Android Wireless Application Development
Developer's Library
Jason Ouellette
Development with the Force.com Platform
Building Business Applications in the Cloud
Developer's Library
Erica Sadun
Second Edition
The iPhone Developer's Cookbook
Building Applications with the iPhone 3.0 SDK
Developer's Library
Addison-Wesley

Visit **developers-library.com** for a complete list of available products

The **Developer's Library Series** from Addison-Wesley provides practicing programmers with unique, high-quality references and tutorials on the latest programming languages and technologies they use in their daily work. All books in the Developer's Library are written by expert technology practitioners who are exceptionally skilled at organizing and presenting information in a way that's useful for other programmers.

Developer's Library books cover a wide range of topics, from open-source programming languages and databases, Linux programming, Microsoft, and Java, to Web development, social networking platforms, Mac/iPhone programming, and Android programming.

PEARSON

Addison-Wesley Cisco Press EXAM/CRAM IBM Press QUE PRENTICE HALL SAMS | Safari Books Online

jQuery, jQuery UI, and jQuery Mobile

Recipes and Examples

Adriaan de Jonge

Phil Dutson

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • 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 authors 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.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact

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

For sales outside the United States, please contact

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Cataloging-in-Publication Data is on file with the Library of Congress.

Copyright © 2013 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: 978-0-321-82208-6
ISBN-10: 0-321-82208-0
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, November 2012

Editor-in-Chief
Mark Taub

Acquisitions Editor
Laura Lewin

Development Editor
Sheri Cain

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Copy Editor
Octal Publishing, Inc.

Indexer
Ted Laux

Proofreader
Diana Coe

Technical Reviewers
Joseph Annuzzi
Jacob Seidelin

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
LaurelTech

❖

To my loving and supportive girlfriend, Christine Kam.
—Adriaan

To my mom, who taught me to love books; my dad, who taught me to love technology; and my family, who supports my indulgence of both.
—Phil

❖

This page intentionally left blank

Contents at a Glance

Preface **xix**

Acknowledgments **xxiii**

About the Author **xxv**

I: Core 1

1 Getting Started with jQuery **3**

2 Selecting Elements **27**

3 Modifying the Page **41**

4 Listening and Responding to Events **65**

5 Communicating with the Server **81**

II: UI 101

6 Interacting with the User **103**

7 Interacting with Widgets **135**

III: Mobile 183

8 Changing the Look and Feel **185**

9 Navigating Pages by Using jQuery Mobile **203**

10 Interacting with jQuery Mobile **249**

IV: Plugins 291

11 Creating Plugins **293**

12 Working with Third-Party Plugins **323**

Index **351**

This page intentionally left blank

Contents

Preface **xix**

Acknowledgments **xxiii**

About the Author **xxv**

I: Core 1

1 Getting Started with jQuery 3

Recipe: Introducing the Basic Usage of jQuery 3

Recipe: Using jQuery with Other Libraries 6

Recipe: Determining the jQuery Version 7

Recipe: Iterating Arrays with each() 8

Recipe: Manipulating Arrays by Using map() 9

Recipe: Working with Arrays of Elements 10

Recipe: Getting the Position of an Element by Using index() 12

Recipe: Finding Elements in an Array by Using grep() 13

Recipe: Determining the Size of an Element Set by Using length() 14

Recipe: Retrieving HTML5 data- Attributes 15

Recipe: Storing Element Data by Using data() 16

Recipe: Removing Element Data by Using removeData() 18

Recipe: Testing and Manipulating Variables 19

Recipe: Extending Objects by Using extend() 22

Recipe: Serializing the Data in a Form 24

Recipe: Testing Browsers for Feature Support 25

Summary 26

2 Selecting Elements 27

Recipe: Combining Two Element Sets by Using add() 27

Recipe: Refining the Selection by Using filter() 28

Recipe: Selecting Descendants by Using find() and children() 29

Recipe: Selecting Elements by Using has() and Testing by Using is() 31
Recipe: Selecting Form Elements by Using Pseudo-Selectors 32
Recipe: Nesting Selectors 33
Recipe: Emulating a CSS Hover Selector 34
Recipe: Selecting on Text by Using contains() 35
Example: Highlighting a Single Word 36
Recipe: Creating Custom Selectors 37
Recipe: Scoping the Selection Context 39
Summary 40

3 Modifying the Page 41
Recipe: Adding Classes 41
Example: Removing Classes 42
Recipe: Generating Class Names 43
Recipe: Changing Properties and Attributes 45
Recipe: Changing the HTML Contained within an Element 46
Recipe: Adding Content by Using append() and appendTo() 48
Example: Appending the Outcome of a Function 49
Recipe: Adding Content by Using prepend() and prependTo() 50
Recipe: Generating HTML Dynamically by Using jQuery 51
Recipe: Attaching and Detaching Elements 53
Recipe: Copying Elements by Using clone() 54
Recipe: Inserting Elements at Specific Positions 56
Example: Moving Items Up and Down the List 58
Recipe: Removing Elements 60
Recipe: Wrapping and Unwrapping Elements 61
Summary 64

4 Listening and Responding to Events 65
Recipe: Listening to Mouse Events 65
Example: Drawing on a Canvas 67
Recipe: Listening to Keyboard Events 69
Recipe: Listening to Form Events 70

Recipe: Listening to Scroll Events 72
Recipe: Adding Central Event Listeners by Using live() and die() 73
Recipe: Delegating Events to Specific Ancestor Elements by Using delegate() 75
Recipe: Changing the Execution Context of a Function by Using proxy() 78
Summary 80

5 Communicating with the Server 81
Recipe: Setting Up an Example Server in Node.js 81
Recipe: Performing a GET Request 83
Recipe: Loading HTML Directly 85
Recipe: Handling the Result by Using Promises 86
Recipe: Handling Server Errors 87
Recipe: Catching Page-not-Found Results 89
Recipe: Handling Page Redirects 90
Recipe: Setting Request Timeouts 91
Recipe: Passing HTTP Headers 93
Example: Validating Form Input on the Server Side 94
Recipe: Loading XML 96
Recipe: Listening to AJAX Events 97
Recipe: Reading JSONP from an External Server 98
Summary 100

II: UI 101

6 Interacting with the User 103
Downloading and Installing jQuery UI 103
Recipe: Dragging Elements 104
    Styling the Draggable Element 105
    Setting Draggable Options 105
    Catching Draggable Events 109
    Invoking Methods on Draggable 111
Recipe: Dropping Elements 111
    Styling the Droppable Element 113
    Setting Droppable Options 113

Catching Droppable Events 114
Invoking Methods on Droppable 115
Recipe: Changing the Order of Elements by Using Sortable 116
Styling a Sortable 117
Setting Sortable Options 117
Catching Events on Sortable 118
Invoking Methods on Sortable 122
Example: Sorting Elements in a Tree Structure 123
Recipe: Selecting Elements in an Unordered List 124
Styling Selectable Elements 126
Setting Selectable Options 126
Catching Events on Selectable 126
Invoking Methods on Selectable 128
Example: Selecting Elements in a Tree Structure 128
Recipe: Resizing Elements 130
Styling the Resizable Element 131
Setting Resizable Options 131
Catching Events on Resizable 133
Invoking Methods on Resizable 134
Summary 134

7 Interacting with Widgets 135
Recipe: Grouping Content by Using the Accordion 135
Styling the Accordion 136
Setting Accordion Options 137
Catching Events on Accordion 139
Methods 141
Recipe: Suggesting Input Values by Using Autocomplete 142
Styling the Autocomplete Component 143
Setting Autocomplete Options 144
Catching Events on Autocomplete 145
Invoking Methods on Autocomplete 147
Recipe: Transforming Elements into Buttons 147
Styling Buttons 149
Setting Button Options 149

Catching Events on Buttons 150
Methods 150
Example: Styling Radio Buttons by Using a Buttonset 151
Recipe: Choosing Dates by Using the Datepicker 152
Styling the Datepicker 152
Setting Datepicker Options 154
Catching Events on Datepicker 158
Invoking Methods on the Datepicker 160
Recipe: Attracting Attention with the Dialog 160
Styling the Dialog 161
Setting Dialog Options 162
Catching Events on the Dialog 164
Invoking Methods on the Dialog 167
Recipe: Displaying Status of a Process by Using the Progressbar 168
Styling the Progressbar 169
Setting Progressbar Options 169
Catching Progressbar Events 169
Invoking Methods on the Progressbar 170
Recipe: Retrieving Numbers by Using a Slider 170
Styling the Slider Element 172
Setting Slider Options 172
Catching Slider Events 173
Invoking Methods on the Slider 174
Recipe: Navigating a Page by Using Tabs 174
Styling the Tab Element 176
Setting Options on the Tabs Widget 177
Catching Events on the Tabs Component 178
Invoking Methods on the Tabs Component 180
Summary 181

III: Mobile 183

8 Changing the Look and Feel 185
Changing the Styling of jQuery UI Components 185
Creating Animations by Using jQuery Core 186

Recipe: Animating Colors by Using jQuery UI 187
Recipe: Hiding Elements by Using Fade and Slide in jQuery Core 189
Recipe: Adding Graphical Effects by Using jQuery UI 191
Recipe: Animating Addition and Removal of CSS Classes by Using jQuery UI 192
Recipe: Displaying all Easing Functions in jQuery UI as Charts 194
Recipe: Displaying All Icons in jQuery UI 196
Recipe: Executing Code at the End of an Animation with a Promise 198
Recipe: Executing Code Within Animations by Using Queue and Dequeue 200
Summary 202

9 Navigating Pages by Using jQuery Mobile 203

Recipe: Setting Up jQuery Mobile Basics 203
Recipe: Serving Multiple Pages from a Single HTML File 205
Recipe: Changing the Title Element 206
Recipe: Loading External Pages Using AJAX 208
Recipe: Displaying Page Loading Messages 210
Recipe: Linking to External Pages without AJAX 212
Recipe: Specifying the Transition Animation 213
Recipe: Specifying Custom Transition Animations 214
Recipe: Listening to Mobile Events 217
Recipe: Generating Back Buttons 221
Recipe: Adding Footers to the Bottom of the Page 223
Recipe: Fixing the Footer Across Pages 225
Recipe: Keeping the Footer in a Fixed Position 227
Recipe: Hiding and Showing the Footer 229
Recipe: Optimizing Headers and Footers for Fullscreen View 231
Recipe: Changing Color Schemes with Themes 232
Recipe: Using Multiple Columns 234
Recipe: Changing Pages by Using JavaScript Calls 236
Recipe: Loading Pages by Using JavaScript Calls 239

Recipe: Attaching Data to DOM Nodes in jQuery Mobile 241
Recipe: Utilizing jQuery Mobile Helper Functions 242
Summary 246

10 Interacting with jQuery Mobile 249
Recipe: Displaying Toolbars Inline in Content 249
Recipe: Displaying a Warning Bar 250
Recipe: Adding Menu Bars to Footer Elements 252
Recipe: Navigating with a Navigation Bar 253
Recipe: Showing and Hiding Elements by Using Collapsible 257
Recipe: Adding Accordion Behavior by Using Collapsible Sets 258
Recipe: Acquiring Basic Text Input by Using Form Fields 260
Recipe: Acquiring Calendar-Based Input 261
Recipe: Displaying Input Fields by Using Alternative Keyboards 263
Recipe: Displaying Specialized Input Fields 265
Recipe: Acquiring Integers by Using Sliders 267
Recipe: Setting Binaries with Flip Switches 268
Recipe: Selecting a Single Element by Using Radio Buttons 269
Recipe: Selecting Multiple Elements by Using Check Boxes 271
Recipe: Selecting Elements from Drop-Down Lists 273
Recipe: Displaying Native Forms by Using jQuery Mobile Off 276
Recipe: Displaying Lists of Elements 278
Recipe: Filtering List Elements 286
Recipe: Grouping Form Elements in Lists 288
Summary 289

IV: Plugins 291

11 Creating Plugins 293
Recipe: Setting Up a Simple Static Plugin 293
Recipe: Creating Simple Context-Aware Plugins 295

Recipe: Chaining jQuery Functions and Plugins 296
Recipe: Parameterizing Plugins 297
Recipe: Providing Default Parameters 299
Recipe: Operating the Plugin by Using Methods 300
Recipe: Creating a Plugin that Creates Plugins 303
Recipe: Registering and Invoking Callback Functions 306
Recipe: Passing Context to Callback Functions 308
Recipe: Returning Deferred Objects to Call Separate Callbacks on Success and on Error 309
Recipe: Returning a Promise to Protect Internals 311
Recipe: Demonstrating the Promise(d) Protection 312
Recipe: Using Promise to Control Flow Structures 314
Recipe: Visualizing Progress Before the Final Callback 315
Recipe: Providing Context to Callback Functions 317
Recipe: Providing a Context to Progress Functions 319
Summary 320

12 Working with Third-Party Plugins 323
Recipe: Displaying a Modal 323
Recipe: Using a Drop-Down Menu 326
Recipe: Using ScrollSpy 328
Recipe: Toggling Tabs 331
Recipe: Adding Tooltips 333
Recipe: Adding a Popover 334
Recipe: Alerting the User 336
Recipe: Button Control 338
Recipe: Collapsing Content 341
Recipe: Putting Content in a Carousel 344
Recipe: Using typeahead for Autocomplete 347
Summary 349

Index 351

Preface

When JavaScript first hit the Internet scene, it was quickly disregarded by experts as a toy suitable for diversion and useless for user interaction. Today, it is the driving force that helps create rich user interfaces, seamless data integration, and client-side support.

Although everything is possible using plain JavaScript, many have discovered the use of various libraries and frameworks to help them do more. jQuery is the perfect library to fill the need for JavaScript integration. The motto of jQuery is *write less, do more.* This is something the jQuery team takes very seriously. Whether you are working on a standard desktop site that needs a little DOM modification, or you're adding your own custom set of controls, or even if you're handling mobile devices, the jQuery team has a library that has been handcrafted and tested on as many platforms, browsers, and devices as possible to ensure the very best experience for both the developer and the user.

Why a Recipe and Example Book on jQuery?

Some have been quick to say that if you know CSS, you know how to use jQuery, but this isn't entirely true. Although having this knowledge certainly helps one to understand the selector engine in jQuery, there are so many extra functions to help manipulate, calculate, and add interaction to your site that a book is simply a must-have to ensure that you are aware of the amazing things that jQuery can do. A recipe and example book is essential because it not only informs you about the functions that you can use, but it also gives you self-contained examples that work with quick explanations to point out the tricky parts or confusing areas.

The inclusion of the jQuery UI and jQuery Mobile libraries makes this book a superior reference guide; now you can easily learn about the other libraries that the jQuery team has made and see how they can fit together to make your next project a seamless, crowd-pleasing success.

Who This Book Is For

Those who have a working knowledge of HTML, CSS, and basic JavaScript should feel at home picking this book up and starting to work through the examples. Those not familiar with Web development might need a little extra time to get up to speed with what is presented here. That having been said, those who learn well by seeing the full HTML page layout with the required script to make the example run will learn quickly and efficiently.

How to Use This Book

Each recipe and example is a set of self-contained scripts that can be loaded onto your favorite Web server and accessed from your browser. Although it is possible to run some of the examples by dropping or loading the example code directly in your browser, any examples that use AJAX will fail, so using a Web server is highly recommended.

When finished reading this book, you should

- Be comfortable adding jQuery to your Web projects
- Understand the differences among jQuery core, jQuery UI, and jQuery Mobile
- Be familiar with the basic functionality and functions of jQuery core
- Be proficient adding styles and widgets, and interacting with the user through jQuery core and jQuery UI
- Be able to create a mobile site by using the jQuery Mobile library
- Be comfortable creating your own plugin and incorporating it into your site
- Be able to add third-party plugins such as those from the Twitter Bootstrap framework

Book Structure

Chapters 1 through 5 cover the jQuery core library. Chapter 1 gives an introduction to the basic usage and how to include jQuery along with setting up arrays, using data attributes, and testing browser feature support. Chapter 2 covers how to fine-tune your element selection from the DOM. Chapter 3 teaches how jQuery can help manipulate the DOM structure of your site. Chapter 4 takes a tour of event handling with jQuery and covers the difference between binding in jQuery versions 1.6 and 1.7. Chapter 5 shows you how to communicate with a Web server, including working with AJAX, page errors, page redirects, and XML.

Chapters 6 through 8 cover the usage of the jQuery UI library. Chapter 6 demonstrates the additional functionality that jQuery UI brings, such as draggable, droppable, and sortable objects. Chapter 7 covers using the widgets, including accordions, auto-complete, buttons, date pickers, dialogs, progress bars, sliders, and tab components. Chapter 8 takes on the styles and theme of the jQuery UI along with effects and transitions.

Chapters 9 and 10 give a course on using the jQuery Mobile library. Chapter 9 starts with the basics of setting up a mobile site and covers page structure, page loading, and page transition and animation. Chapter 10 covers the additional widgets and styles that jQuery Mobile provides, including navigation bars, sliders, flip switches, and form elements.

Chapters 11 and 12 cover the use of plugins. Chapter 11 provides a comprehensive overview of what a plugin consists of and how to create your own with method handling and function chaining. Chapter 12 gives examples of using trusted third-party plugins from the Twitter Bootstrap framework to add extra functionality and style to your project.

Additional Resources

There are many places that you can learn more about jQuery, jQuery UI, and jQuery Mobile. The official sites themselves offer excellent documentation and explain the choices behind deprecating code and give hints and warnings about upcoming features.

Here are some helpful sites that you should visit:

- jQuery API docs: http://api.jquery.com/
- jQuery UI docs: http://jqueryui.com/demos/
- jQuery Mobile docs: http://jquerymobile.com/demos/
- Twitter Bootstrap JavaScript docs: http://twitter.github.com/bootstrap/javascript.html
- Zurb Foundation docs: http://foundation.zurb.com/docs/
- jQuery Tools: http://jquerytools.org/
- Stack Overflow forums: http://stackoverflow.com/

This page intentionally left blank

Acknowledgments

First of all, I would like to thank Phil Dutson for joining this book project and helping to finish it before the deadline. In addition to Laura Lewin, Olivia Basegio, Sheri Cain, Jacob Seidelin, and Joseph Annuzzi—as mentioned by Phil—I would also like to thank Trina MacDonald, who started this book project at Pearson.

I would like to say thanks to my colleagues, both at my former employer, ANWB, and my new employer, Xebia, for giving me inspiration and great ideas. I love working with people who share the same interest in new technologies!

I am thankful for my family and friends who encourage me wherever I go. And in particular, I am happy and blessed with my girlfriend, Christine Kam. Thank you for your unconditional support!

—Adriaan

I'd like to thank Adriaan de Jonge who started this project and was gracious enough to allow me to help him finish it. I also want to recognize all of the hard work and countless hours put into this project by the amazing team at Pearson. Thanks go to Laura Lewin who brought me onto the project; Olivia Basegio for coordinating everything; Sheri Cain for correcting the flow and helping structure the visual representation of the material; my absolutely brilliant tech editors Joseph Annuzzi and Jacob Seidelin, who not only made sure that the code runs properly, but that the explanations about why it runs were clear and accurate. Thanks also go to Bob Russell of Octal Publishing, Inc., for his commendable copyediting prowess and Julie Nahil at Pearson for all of the coordination during the production process.

I also want to thank my designer friends from UltraCube (Alan Lawrence, Jake Stuart, Candis Sorenson, Kaylee White, and Jen Lewis) who constantly remind me that life is whatever you make of it, and my current e-commerce team (Tracy Cox, Brett Wood, Ryan Jensen, Remo Galli, Eric Barnett, and Chris Lewis) who remind me to always make sure you are doing the things you love.

—Phil

This page intentionally left blank

About the Authors

Adriaan de Jonge works as a Consultant for Xebia IT Architects in The Netherlands. Adriaan specializes in Internet, content management, and Java. He is interested in new technologies such as MongoDB, Node.js, HTML5, and various cloud-computing platforms. Adriaan is also the author of *Essential App Engine* (Addison-Wesley, 2012) and several articles for IBM developerWorks on Java, XML, and Internet technologies. He has experience as a conference speaker at JFall 2011 and the Scandinavian Developer Conference in 2012.

Phil Dutson is the lead front-end developer for ICON Health and Fitness. He has worked on projects and solutions for NordicTrack, ProForm, Freemotion, Sears, Costco, Sam's Club, and others. He was an original team member of the iFit team that integrated Google Maps into personalized workout creation and playback. Phil co-founded and currently manages *The E-Com DevBlog*, a development blog focused on Web development and solutions. To learn more visit http://dev.tonic1394.com. He is also the author of *Sams Teach Yourself jQuery Mobile in 24 Hours* (Sams, 2013).

This page intentionally left blank

Chapter 5

Communicating with the Server

This chapter discusses how to communicate with the server. To demonstrate this, a simple test server is written in Node.js, which is a server-side JavaScript execution environment. After that, various recipes show how to get JSON, HTML, XML, and JSONP from the server by using AJAX methods and functions. This chapter pays a lot of attention to error handling to assure the quality of the end result.

Recipe: Setting Up an Example Server in Node.js

To test the AJAX examples, you need a server. For the examples, it does not matter whether the server is written in PHP, Python, Ruby, Erlang, Dart, Go, .Net, or Java. Keep in mind, of course, that all servers take some setup time.

Because jQuery is JavaScript, this book assumes that you are already familiar with it. That is why the test server is provided in JavaScript code. To run the code, you need to download and install Node.js at http://nodejs.org.

There are no plugins or further modules needed for this example. For reference, the code in this chapter was developed and tested by using versions 0.4.11 and 0.6.19 of Node.js. After installing Node.js and putting the example in a file called 01-app.js, you can run the code from the following command line:

```
node 01-app.js
```

If this command does not start your node server, ensure that you have correctly added the node to your system path. When running the code, you can access subsequent examples at http://localhost:1337/02-ajax-get.html.

Listing 5.1 contains the implementation for the Node.js test server.

Listing 5.1 **Listening for HTTP Requests Generated by Recipes in This Chapter and Responding Accordingly**

```
var http = require('http'),
  url = require('url'),
  fs = require('fs');
http.createServer(function (req, res) {
  var reqData = {
    url: url.parse(req.url, true),
    method: req.method,
    headers: req.headers },
  path = reqData.url.pathname;

  if(path.match(/^\/[0-9a-z\-]+\.(html)|(json)|(xml)$/))
    fs.readFile('.' + path, function (err, data) {
      if (err) {
        res.writeHead(404, {'Content-Type': 'text/plain'});
        res.end('not found');
      }
      else {
        if(path.split('.')[1] == 'html')
          res.writeHead(200, {'Content-Type': 'text/html'});
        else if(path.split('.')[1] == 'xml')
          res.writeHead(200, {'Content-Type': 'application/xml'});
        else
          res.writeHead(200, {'Content-Type': 'application/json'});
        res.end(data);
      }
    });
  else if(path == '/return-http-headers') {
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end(JSON.stringify(reqData));
  }
  else if(path == '/sleep') {
    var endTime = new Date().getTime() + 2000;
    while (new Date().getTime() < endTime);
    res.writeHead(500, {'Content-Type': 'text/plain'});
    res.end('slow response');
  }
  else if(path == '/validate') {
    var keys = [];
    for(var key in reqData.url.query) {
      if(reqData.url.query[key] == '')
        keys.push(key);
    }
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end(JSON.stringify(keys));
```

Listing 5.1 **Listening for HTTP Requests Generated by Recipes in This Chapter and Responding Accordingly (Continued)**

```
  }
  else if(path == '/redirect') {
    res.writeHead(302, {
      'Location': '/test-values.json' });
    res.end();
  }
  else if(path == '/fail\-on\-purpose') {
    res.writeHead(500, {'Content-Type': 'text/plain'});
    res.end('unexpected" error');
  }
  else {
    res.writeHead(404, {'Content-Type': 'text/plain'});
    res.end('not found');
  }
}).listen(1337, "localhost");
console.log('Server running at http://localhost:1337/');
```

HTML, JSON, and XML requests are passed to the file server. Some special cases you will encounter in the following recipes are handled each with their own specific response. Given the requested path, the responses should not be too surprising.

If the file cannot be found, or there is no handler and the request is not XML, JSON, or HTML, a 404 error is returned.

Recipe: Performing a GET Request

One of the simpler AJAX requests can be executed by using the shorthand method for `GET`. You can imagine performing a similar call for `POST`, `PUT`, and `DELETE`. Keep in mind that `PUT` and `DELETE` are not supported by all browsers, so it is wise to use `GET` and `POST`. Listing 5.2 shows the use of the `get()` method.

Listing 5.2 **Fetching JSON Values by using the get() Shorthand Function**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>The AJAX get() request function</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>
```

(Continues)

Listing 5.2 **Fetching JSON Values by using the get() Shorthand Function (Continued)**

```
09
10 <button id="trigger">GET</button>
11 <br>
12 <div id="target"></div>
13
14 <script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
15
16 <script>
17 // please externalize this code to an external .js file
18 $(document).ready(function() {
19
20   $('#trigger').click(function() {
21
22     $.get('02a-test-values.json', function(data) {
23
24       $('#target').append('The returned value is: ' + data.name);
25
26     }, 'json');
27   });
28
29 });
30 </script>
31 </body>
32 </html>
```

Line 22 fetches the following JSON document:

```
{
  "name": "Adriaan de Jonge",
  "email" : "adriaandejonge@gmail.com"
}
```

When executed, this script returns the name of the author. In this example, the success handler function is a callback directly passed to the `get()` function. The last parameter passed in the `get()` is `'json'`. This is an optional data-type parameter. In this instance, it informs jQuery that the data requested will be returned in JSON format. Later examples demonstrate an alternative approach for callback functions. If you use a web service or have some server-side logic set up to handle data coming in before handing back a response, you can pass an additional set of data to the server through the `get()` function. To learn more about sending data to the server by using `get()`, see the official documentation at http://api.jquery.com/jQuery.get/.

Recipe: Loading HTML Directly

Even simpler than the `get()` method is to use `load()` to gather a snippet of HTML and put it directly into the document. Listing 5.3 demonstrates how to do this.

Listing 5.3 **Filling a <div> with a Dynamic HTML Snippet**

```
00 <!DOCTYPE html>
01
02 <html lang="en">
03 <head>
04   <title>The AJAX load() request</title>
05 </head>
06 <body>
07
08 <h2>Press the button to perform the request.</h2>
09
10 <button id="trigger">GET</button>
11 <br>
12 <div id="target">
13
14 <script src="http://code.jquery.com/jquery-latest.min.js"></script>
15
16 <script>
17 // please externalize this code to an external .js file
18 $(document).ready(function() {
19
20   $('#trigger').click(function() {
21
22     $('#target').load('03a-test-snippet.html');
23
24   });
25
26 });
27 </script>
28 </body>
29 </html>
```

The HTML that will be loaded into the document is as follows:

```
<h1>Hello world from external HTML snippet</h1>
```

Line 22 demonstrates how the `load()` function works directly on a selection of elements without requiring a callback handler.

Recipe: Handling the Result by Using Promises

In Listing 5.2, you saw a callback handler being passed to the `get()` method to be called after a successful result from the AJAX request. The advantage of callbacks is asynchronous execution of code. The browser remains responsive while the AJAX request is working in the background. However, when you have many callbacks, the code starts looking like a Christmas tree. It is a callback inside a callback inside a callback. And what about failures?

Promises can solve this problem. Understanding promises involves some theory. First, let's explore how they work by looking in Listing 5.4.

Listing 5.4 **Demonstrating done(), fail(), and always()**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>The done() fail() always() function</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">

<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external.js file
$(document).ready(function() {

  $('#trigger').click(function() {

    // avoid error() success() complete()
    // those are deprecated in jQuery 1.8
    // use done() fail() always() instead

    $.ajax({url:' 02a-test-values.json', dataType: 'json'})
    .done (function(data) {
      $('#target').append('The returned value is: '
                            + data.name + '<br>');
    })
    .fail(function() {
      $('#target').append('The AJAX call failed.<br>');
    })
```

Listing 5.4 **Demonstrating done(), fail(), and always() (Continued)**

```
    .always(function() {
      $('#target').append('finished anyway.');
    });
  });
});
</script>
</body>
</html>
```

Notice in this script that chained functions provide callbacks after calling the `ajax()` function. This is possible because `ajax()` returns a promise object. A promise is a safe version of a *deferred* object. Deferred objects are discussed in Chapter 11, "Creating Plugins." The difference is that a promise does not expose the internals of a deferred. Then, what does a promise do?

When you call the `done()` function with a callback handler and the `ajax()` request has not yet returned, the callback is stored in a queue. Once the `ajax()` request returns and the result is successful, all the queued callbacks under `done()` are called. When you call the `done()` function with a callback handler after the `ajax()` request has returned, if the result was successful, the callback is executed right away.

Callbacks passed to `fail()` and `always()` are handled similarly. Only `fail()` executes when the result was unsuccessful, of course.

There is a shorthand notation for `done()` and `fail()` called `then()`. The `then()` function takes two arguments: one callback in case of success and one in case of failure. The following snippet shows how you can use it to replace the `done()` and `fail()` methods used in Listing 5.4:

```
$.ajax({url:'02a-test-values.json', dataType: 'json'})
    .then(
      function(data) {
        $('#target').append('The returned value is: '
                          + data.name + '<br>');
      },
      function() {
        $('#target').append('The AJAX call failed.<br>');
      }
    );
```

Recipe: Handling Server Errors

To test the `fail()` handler, the test server from the first recipe provides a `fail-on-purpose` URL. Listing 5.5 calls this URL to see what happens.

Listing 5.5 **Catching Server Errors by Using fail()**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Test case: failure</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

    $.ajax('fail-on-purpose')
    .done(function(data, xhr) {
      $('#target').append('Unexpected success. . . ' +
                     '(actually not a good thing)');
     })
    .fail(function(xhr, text, error) {
      $('#target').append('Failed as expected (good!). Code ' +
                            xhr.status + ' and text ' + error);
    });

  });

});
</script>
</body>
</html>
```

As expected, the `fail()` callback is called. From the parameters passed to this callback handler, you can determine what went wrong and act accordingly.

Recipe: Catching Page-not-Found Results

Similarly, Listing 5.6 tests what happens when the page is not found.

Listing 5.6 **Recognizing a Page-not-Found Code**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Test case: page not found</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

    $.ajax('not-found')
    .done(function(data, xhr) {
      $('#target').append('Unexpected success… ' +
                '(actually not a good thing)');
    })
    .fail(function(xhr, text, error) {
      $('#target').append('Failed as expected (good!). Code ' +
      xhr.status + ' and text ' + error);
    });

  });

});
</script>
</body>
</html>
```

As expected, again, the `fail()` callback is called, only this time with different parameters. Although you could check the `xhr.status` for a specific error code, jQuery provides a setting in `ajax()` that looks specifically at HTTP codes. The following snippet of code could be used in the previous example, replacing line 23.

```
$.ajax('fail-on-purpose', {
    statusCode: {
        404: function() {//insert 404 handling function here},
        500: function() {//insert 500 handling function here}
    }
})
```

When you remove the comments and insert your own code, the code will be run when either a `404` or `500` HTTP status code is returned. If you leave the `fail()` method in place, it will also execute, giving you another error handling point.

Recipe: Handling Page Redirects

You might expect a similar result when you encounter a redirect code. Listing 5.7 investigates what happens in this case.

Listing 5.7 **Receiving Content After an Implicit Redirect**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Test case: redirect</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

```

Listing 5.7 **Receiving Content After an Implicit Redirect (Continued)**

```
    $.ajax('redirect')
    .done(function(data, xhr) {
      $('#target').append('Successfully redirected. . . ' +
                     'returned data is ' + data.name);
    })
    .fail(function(xhr, text, error) {
      $('#target').append('Redirect failed. Code ' +
                     xhr.status + ' and text ' + text);
    });

  });

});
</script>
</body>
</html>
```

In this case, the AJAX request is successfully redirected to the new URL. The resulting values are picked up by the `done()` handler. In this example, this is exactly what we want.

However, if you do not want to be redirected and know that something is wrong, you might be out of luck. It seems like jQuery has nothing to do with the automated redirect handling. The browser does this for you according to specification.

In some use cases, a redirect is used to ask for further details from the end user, such as login credentials. In these cases, you do not want the AJAX request to be redirected under the hood. Instead, you want to catch the redirect and pass it on to the full browser screen.

Under these circumstances, if you have no control over the server, you are out of luck. If you do have control over the server, you should find a different way to pass the direct back to the AJAX call. Consider making it part of the content payload. Some libraries have invented new non-standard HTTP status codes. That should be considered *bad practice* and can cause unexpected results with proxies and caches.

Recipe: Setting Request Timeouts

In a responsive interface, it is sometimes better to quickly say that you cannot answer in time than it is to wait for a long time. If the user is waiting for the results of an AJAX request, it is wise to add a timeout.

Listing 5.8 tries to fetch the result of the `sleep` function in one second. A quick glance at the first recipe in this chapter teaches us that the test server will never respond before two seconds have passed. You do not have to be psychic to predict the outcome here. . . .

Listing 5.8 **Failing If the Server Takes Longer Than One Second**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>The timeout property</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

    $.ajax({url: 'sleep',
        timeout : 1000
    })
    .done(function(data, xhr) {
      $('#target').append('Response in time after all: ' + data);
    })
    .fail(function(xhr, text, error) {
      $('#target').append('Failed as expected: ' + error);
    });

  });

});
</script>
</body>
</html>
```

Once again, the `fail()` callback is called. And once again, it's called with different parameters, describing that this failure was caused by a timeout.

Recipe: Passing HTTP Headers

If you need control over HTTP headers—for example, those used for caching—you can pass anything with the AJAX request. To test how this works, the test server from the first example returns a JSON string containing exactly the headers it received during the request.

Listing 5.9 passes a simple ETag header and displays the result returned from the test server.

Listing 5.9 **Passing an ETag Header and Displaying the Returned Mirrored Headers from the Server**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>The headers property</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

    $.ajax({url: 'return-http-headers',
        headers : {
            ETag: '12345'
      }
    })
    .done(function(data, xhr) {
      $('#target').append(
        $.map(data.headers, function(i, name) {
            return name + ' = '+ data.headers[name] + '<br/>';
```

(Continues)

Listing 5.9 **Passing an ETag Header and Displaying the Returned Mirrored Headers from the Server (Continued)**

```
          })
         .join(' ')
      );
    })
    .fail(function(xhr, text, error) {
      $('#target').append('Failed unexpectedly');
    });
  });

});
</script>
</body>
</html>
```

The test server returns a little bit more than just the HTTP headers, which might be useful for different tests such as ETag configuration, compression support (gzip, deflate), and even agent identification.

Example: Validating Form Input on the Server Side

In Chapter 1, "Getting Started with jQuery," you saw the `serialize()` function. Chapter 4, "Listening and Responding to Events," demonstrated how to catch events on form elements. Combine these two with an AJAX request, and you can validate the input of a form before it is submitted.

The test server from the first recipe returns all the input element names that send an empty string. Listing 5.10 adds a red border to all element names returned by the validation function.

Listing 5.10 **Serializing and Sending the Form Input to the Server After Every Change**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Use serialize() for side field validation</title>
  <style>
 /* please externalize this code to an external
    .css file */
    .error {
      border-color: red;
    }
```

Listing 5.10 **Serializing and Sending the Form Input to the Server After Every Change (Continued)**

```
  </style>
</head>
<body>

<h2>Fill out the fields and see the result.</h2>

<form action="" method="post">
  <label for="first_field">First field</label>
  <input type="text" name="first_field"
    value="" id="first_field"><br>

  <label for="second_field">Second field</label>
  <input type="text" name="second_field"
    value="" id="second_field"><br>

  <label for="third_field">Third field</label>
  <input type="text" name="third_field"
    value="" id="third_field"><br>

  <label for="fourth_field">Fourth field</label>
  <input type="text" name="fourth_field"
    value="" id="fourth_field"><br>

  <input type="submit" name="submit" value="Submit" id="submit">

</form>

<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('input').filter(':text').addClass('error');

  $('input').change(function() {

    $.get('validate', $('form').serialize())
    .done(function(data) {

      $('input').removeClass('error');

      $.each(data, function(i, name) {
```

(Continues)

Listing 5.10 **Serializing and Sending the Form Input to the Server After Every Change (Continued)**

```
        $('#' + name).addClass('error');
      });

    });

  });

});
</script>
</body>
</html>
```

In a real-world example, you could imagine a more sophisticated validation function. Even if you can validate many things on the client by using JavaScript, good practice is to validate again on the server upon final submission. (Consider that a client can be easily replaced by rogue code in transit.) Having JavaScript code on both the client and the server, you can invent a scenario where you can reuse parts of the validation code.

Recipe: Loading XML

The original meaning of what was once the *acronym* AJAX was Asynchronous JavaScript And Xml. The XML part of this name slowly disappeared over time as HTML and JSON have become more popular alternatives (and thus, AJAX is technically no longer an acronym).

If you still want to communicate with the server by using XML, you certainly can. Listing 5.11 demonstrates that jQuery even makes it easy for you to read the XML. It works similarly to selecting HTML elements.

Listing 5.11 **Reading XML Values Returned by the Server by Using jQuery**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Get XML from server</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
```

Listing 5.11 **Reading XML Values Returned by the Server by Using jQuery (Continued)**

```
12 <div id="target">
13
14
15 <script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
16
17 <script>
18 // please externalize this code to an external .js file
19 $(document).ready(function() {
20
21   $('#trigger').click(function() {
22
23     $.get('11a-test-values', function(data) {
24
25       $('#target').append('The returned value is: ' +
26          $(data).find('name').text());
27     });
28
29   });
30
31 });
32 </script>
33 </body>
34 </html>
```

Line 26 uses jQuery constructions to read from the XML. The following snippet contains the XML returned by the server:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <name>Adriaan de Jonge</name>
   <email>adriaandejonge@gmail.com</email>
</root>
```

Imagine working with a larger XML example. Using jQuery functions such as those that have been shown, it remains manageable.

Recipe: Listening to AJAX Events

Similar to mouse, keyboard, and scroll events, in jQuery, AJAX requests generate AJAX events. You can use these to generate status indicators on screen so that the visitor knows that work is in progress. Listing 5.12 shows a basic event log for AJAX events.

Listing 5.12 **Displaying all AJAX Events**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>AJAX related event handlers</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target"> </div>
<div id="log"> </div>

<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {
    $('#target').load('test-snippet.html');
  });

  $.each(('ajaxError ajaxSend ajaxStart ajaxStop ' +
          'ajaxSuccess ajaxComplete').split(' '),
    function (i, name) {
      $('#log').bind(name, function(event, xhr) {
      $(this).append('Event: ' + event.type + '<br/>');
    });
  });
});
</script>
</body>
</html>
```

Watch the order in which the events arrive when you click the button. This means you can respond in several stages of the AJAX request. The events also allow you to modify or add request parameters if you need to from a central location.

Recipe: Reading JSONP from an External Server

Classic AJAX works with the `XmlHttpRequest (XHR)` object. Most browsers do not allow XHR to access other servers than the origin of the current page. To work around this limitation, JSONP was invented.

JSONP is JSON wrapped inside a function call. Instead of making an AJAX request, a `script` element pointing to the JSONP script is added inside the HTML document and a callback function is called to access the script.

Caution

The browser vendors who restricted the use of XHR had good reason to do so: security. Working around these restrictions with JSONP opens up new possibilities for hackers because JSONP does not allow validation before execution.

Listing 5.13 demonstrates how to retrieve data from Twitter by using JSONP. Keep in mind that error handling does not work with JSONP.

Listing 5.13 **Connecting to Twitter and Searching for jQuery-Related Posts**

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Get JSONP</title>
</head>
<body>

<h2>Press the button to perform the request.</h2>

<button id="trigger">GET</button>
<br>
<div id="target">


<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>

<script>
// please externalize this code to an external .js file
$(document).ready(function() {

  $('#trigger').click(function() {

    $.getJSON('http://search.twitter.com/search.json' +
        '?q=jquery&callback=?', function(data) {

      $.each(data.results, function(index, value) {

        $('#target').append(value.text + '<br>');

      });

    });

```

(Continues)

Listing 5.13 **Connecting to Twitter and Searching for jQuery-Related Posts (Continued)**

```
    });

});
</script>
</body>
</html>
```

By default, the JSONP handler in jQuery looks for a `callback=?` parameter in the query string. The `?` is replaced with a jQuery-generated callback function to be inserted into JSONP by the server.

You can modify settings if the parameter has another name than `callback` or when the callback method is not parameterized. If you insist, you can find these parameters in the jQuery online documentation.

The best advice is to avoid JSONP whenever possible.

Summary

This chapter covered the most common uses of AJAX. The recipes cover fetching JSON, HTML, and XML, as well as many error scenarios. You can pass HTTP headers, validate form input in the background, and listen for AJAX events in general. The last recipe showed a dirty hack to load data from other servers by using JSONP. Because of security risks and a lack of error handling, it is better to avoid JSONP.

Index

A

abort() method for tabs, 181
accept option, 113
Accordions
  collapsible sets, 258–260
  events, 139–141
  methods, 141
  options, 137–139
  styling, 136–137
  transforming elements, 135–136
accordion-toggle class, 343
activate event
  droppable areas, 114
  sortable, 121
activate() method, 141
active option, 138
activeClass option, 113
add event, 179
add() function for combining element sets, 27–28
add() method for tabs, 180
addClass() function, 41, 43–45
addClasses option, 106
Addition of CSS classes, animating, 192–194
after() function, 58
AJAX
  linking to external pages without, 212–213
  listening to events, 97–98
  loading external pages, 208–210

ajax() function, 87

ajax property, 26

alert() function, 323

Alerts for users, 336–338

Aliases, dollar signs for, 6–7

allowSamePageTransition option, 238

alsoResize option, 132

Alternative keyboards, 263–265

altField option, 155

altFormat option, 155

always() function, 86–87, 314–315, 317

Ancestor elements, delegating events to, 75–78

animate option
- resizable elements, 132
- sliders, 172

animated option, 138

animateDuration option, 132

Animations
- code execution at end of, 198–200
- code execution using queue and dequeue, 200–202
- colors, 187–189
- creating, 186–187
- CSS class addition and removal, 192–194
- rectangle size and position, 186–187
- transition, 213–216

append() function, 48–49, 330

Appending
- date text, 155
- outcome of functions, 49–50

appendText option, 155

appendTo() function, 48–49, 53–54

appendTo option
- autocomplete, 144
- draggable elements, 106

apply() function, 303, 309

Array elements
- finding, 13–14
- iterating, 8–9
- manipulating, 9–10
- modifying, 19–22
- position of, 12–13
- working with, 10–12

Aspect ratio setting, 132

Attaching
- data to DOM nodes, 241–242
- elements, 53–54

attr() function, 45–46

Attributes vs. properties, 45–46

authority option, 246

Autocomplete
- suggestions, 142
- typeahead for, 347–348

autoFocus option, 145

autoHeight option, 138

autoHide option, 132

autoOpen option, 162

autoRefresh option, 126

autoSize option, 155

axis option
- draggable elements, 106
- sortables, 117

B

Back buttons, 221–223

Background color, 34–35

before() function, 58

beforeClose event, 164

beforeShow event, 159

beforeShowDay event, 159

beforeStop event, 120

Binaries, flip switches for, 268–269

blur event, 72

Boilerplate code for plugins, 303–306
Bootstrap framework, 323, 325
Bottom of page, footers at, 223–224
Bresenham's line algorithm, 69
Browser features, testing for, 25–26
browser property, 26
buttonImage option, 155
buttonImageOnly option, 155
Buttons
alerts, 336–338
back, 221–223
dates, 155–157
footers with, 223–225
for hiding and showing footers, 229–231
for multiple columns, 235–236
radio, 32–33, 269–271
selecting, 32–33
styling lists of elements as, 148–149
third-party plugins, 338–340
transforming elements into, 147–150
up and down, 58–59
buttons option for dialogs, 163
Buttonsets
in footers, 224–225
styling radio buttons using, 151
buttonText option, 155

C

cache option, 177
Calendar-based input, 261–263
Callback functions
calling on success and error, 309–311
chains, 86–87, 200
context for, 308–309, 317–319
invoking, 316–317
registering and invoking, 306–308
visualizing progress before, 315–317
Callback() object, 306–308
cancel() method for sortable, 123
cancel option for draggable elements, 106
canvas element, drawing on, 67–69
carousel-caption class, 346
carousel() function, 346
Carousels for images, 344–346
CDN (Content Delivery Network), 5, 103
Central event listeners, 73–75
Chaining
callback functions, 86–87, 200
functions and plugins, 296–297
change event
accordions, 140
autocomplete, 147
progressbars, 170
purpose, 72
sliders, 174
sortable, 119–120
changeHash option, 238
changeMonth option, 155
changePage() function, 238
changestart event, 141
Charts, displaying easing functions as, 194–196
Check boxes
displaying, 271–273
selecting, 32–33
children() function, 29–30
class attribute, 216
class element, 5
Classes
adding, 41–42
CSS, animating, 192–194

Classes (*contd.*)
names, 43–45
removing, 42–43
clearStyle option, 138
click() function, 338
clone() functions, 54–56
close event
autocomplete, 146
dialogs, 167
closeOnEscape option, 163
closeText option
datepicker, 156
dialogs, 163
Collapsible content, 341–343
collapsible elements, 257–258
collapsible option
accordions, 138
tabs, 177
Collapsible sets, 258–260
Color
animating, 187–189
background, 34–35
input fields, 265–266
scheme changes, 232–234
Columns
multiple, 234–236
splitting lists into, 280–281
Combining element sets, 27–28
complete event, 170
Connecting
to Twitter, 99–100
unordered lists, 123–124
connectToSortable option, 106
connectWith option, 117
constrainInput option, 156
containment option, 107
contains() function, 35–36
Content
adding to pages, 48–51
carousels, 344–346
collapsible, 341–343
grouping, 135–141
hiding, 42–43
toolbars in, 249–250
Content Delivery Network (CDN), 5, 103
Context
to callback functions, 308–309, 317–319
chainability, 296–297
execution, 78–80
to progress functions, 319–320
selection, 39–40
Context-aware plugins, 295–296
cookie option, 177
Copying elements, 54–56
Count items with lists, 283–284
create event
accordions, 140
autocomplete, 145
buttons, 150
datepicker, 158
dialogs, 164
droppable areas, 114
resizable elements, 133
selectable elements, 126
sliders, 173
sortable, 119
tabs, 178
CSS
class animation, 192–194
hover selector emulation, 34–35
within HTML, 4
css() function, 5, 29, 297
CSS3 transitions, 214–216
currentText option, 156
cursor option, 107
cursorAt option, 107
Custom selectors, 37–39
Custom transition animations, 214–216

D

Data
  attaching to DOM nodes, 241–242
  serializing, 24–25
data- attributes, retrieving, 15–16
data-add-back-button attribute, 223
data-ajax attribute, 213
data-content attribute, 336
data-filter attribute, 287
data() function, 16–18
data-iconpos attribute, 256–257
data-inset attribute, 279
data-items attribute, 348
Data option, 238
data-original-title attribute, 336
data-parent attribute, 343
data-position attribute, 227
data-prefetch attribute, 239
data-provide attribute, 348
data-rel attribute, 223
data-role attribute, 225
data-slide attribute, 346
data-source attribute, 348
data-split-icon attribute, 281
data-spy attribute, 330
data-title attribute, 208
data-toggle attribute, 326, 328, 332, 340, 343
data-transition attribute, 214, 216
data-type attribute, 225, 271, 273
Data type testing function, 21
dataUrl option, 238
datepicker
  choosing dates, 152
  events, 158–159
  methods, 160
  options, 154–158
  styling, 152–154
deactivate event
  selectable elements, 128
  sortable, 122
deactive event, 115
Default plugin parameters, 299–300
Deferred construction, 199
Deferred objects
  callbacks for, 306
  promises, 87, 199
  returning, 309–311
delay option
  autocomplete, 144
  draggable elements, 107
delegate() function, 75–78
Delegating events
  central event listeners, 73–75
  to specific ancestor elements, 75–78
Deprecated APIs, 5
dequeue() function, 200–202
Descendants, selecting, 29
destroy() method
  accordions, 141
  dialogs, 167
  dragging elements, 111
detach() function, 53–54
Detaching elements, 53–54
dialog() method, 160
dialogClass option, 163
Dialogs
  events, 164–167
  methods, 167
  options, 162–164
  styling, 161–162
  transforming paragraphs into, 160–161
die() function, 73–75
directory option, 246
disable event for tabs, 180
disable() function, 307–308

disable() method
- accordions, 141
- dialogs, 167
- dragging elements, 111
- tabs, 181

disabled event for buttons, 150

disabled option
- accordions, 138
- autocomplete, 144
- buttons, 149
- datepicker, 154
- dialogs, 162
- draggable elements, 106
- progressbars, 169
- sliders, 172
- tabs, 177

distance option, 107

<div> elements
- accordions from, 135–136
- list ordering, 58–59
- progressbars from, 168–169
- resizable, 130–131
- sliders from, 171

Dollar signs ($)
- as alias, 6–7
- multiple event sources, 79–80

DOM trees
- attaching data to nodes, 241–242
- moving elements around, 53–54

domain option, 246

done() function, 86–87, 91

Down buttons, 58–59

Downloading jQuery UI, 103–104

drag event
- dialogs, 165
- elements, 110

dragcreate event, 109–110

draggable option, 163

Dragging elements, 104–105
- events, 109–110
- methods, 111
- options, 105–109
- styling, 105

dragStart event, 165

dragStop event, 166

Drawing on canvas, 67–69

Drop-down lists, 273–276

Drop-down menus, 326–328

drop event, 115

dropOnEmpty option, 117

Dropping elements, 111–112
- events, 114–115
- methods, 115
- options, 113–114
- styling, 113

Dummy text, 142

duration option, 156

Dynamic HTML
- generating, 51–53
- snippet, 85

Dynamic navigation, 328–331

E

each() function, 8–9

Easing functions
- displaying as charts, 194–196
- list of, 190

effect function, 191–192

Effects, graphical, 191–192

Element selection
- check boxes, 271–273
- descendants, 29–30
- drop-down lists, 273–276
- filtering, 28–29
- form, 32–33
- has() for, 31

list, 148–149
multiple, 271–273
radio buttons, 269–271
single, 269–271
text, 35–36
tree structures, 128–130
unordered lists, 124–130
words, 36–37

Element sets
combining, 27–28
size, 14–15

Elements
array. *See* Array elements
attaching and detaching, 53–54
collapsible, 257–258
copying, 54–56
dragging, 104–111
dropping, 111–115
hiding, 189–191
HTML in, 46–47
inserting, 56–58
lists. *See* Lists
removing, 18–19, 60–61
resizing, 130–134
selecting, 124–130
sortable, 116–123
sorting, 123–124
storing, 16–18
transforming into buttons, 147–150
wrapping and unwrapping, 61–64

empty() function, 60–61

Emulating
CSS hover selectors, 34–35
repetitive elements, 55–56

enable event for tabs, 180

enable() method
accordions, 141
dialogs, 167
dragging elements, 111
tabs, 181

end function, 28–29

Errors
callbacks calling on, 309–311
servers, 87–88

ETag headers, 93–94

Even elements, separating, 28–29

event option
accordions, 138
dragging elements, 110
tabs, 177

Events, 65
accordions, 139–141
AJAX, 97–98
autocomplete, 145–147
buttons, 150
central event listeners, 73–75
datepicker, 158–159
delegating, 75–78
dialogs, 164–167
drawing, 67–69
droppable, 114–115
execution context, 78–80
form, 70–72
keyboard, 69–70
modal window triggers, 324–325
mouse, 65–69
progressbars, 169–170
resizable elements, 133–134
scroll, 72–73
selectable elements, 126–128
sliders, 173–174
sortable, 118–122
tabs, 178–180

Executing code in animations
with promises, 198–200
using queue and dequeue, 200–202

Execution context of functions, 78–80
extend() function, 22–24, 299, 303
Extending
  default parameters, 299–300
  objects, 22–24
External pages
  linking to, 208–210, 212–213
  loading, 208–210
External servers, reading from, 98–100

F

Fade animation, 189–191, 213
fail() function, 86–88, 90, 314–315
filename option, 246
Filler text, 142
fillSpace option, 139
filter() function, 28–29, 33
filter option, 126
Filtering
  list elements, 286–287
  selectable elements, 126
  selections, 28–29
find() function, 29–30
Finding array elements, 13–14
fireWith() function, 308–309
firstDay option, 157
Fixed footers, 225–229
Fixed toolbars, 227–229
flip animation, 213
Flip switches, 268–269
Flow structures, promises for, 314–315
focus event
  autocomplete, 146
  dialogs, 165
  purpose, 72
Footers
  bottom of page, 223–225
  fixed position, 225–229
  fullscreen view, 231–232
  hiding and showing, 229–231
  menu bars for, 252–253
for loops, 9
forceHelperSize option, 118
forcePlaceholderSize option, 118
Form elements
  grouping in lists, 288–289
  selections, 32–33
  serializing, 24–25
Forms
  events, 70–72
  input validation, 94–96
  without jQuery Mobile, 276–277
  repetitive elements, 55–56
  text input fields, 260–261
Fullscreen view for headers and footers, 231–232
Functions
  appending outcome of, 49–50
  callback. *See* Callback functions
  chaining, 296–297
  execution context, 78–80
  parameters, 301–303
  passing, 44–45
fx option, 178

G

Gallery carousels, 344–346
Generating
  class names, 43–45
  HTML dynamically, 51–53
get() function in Ajax, 83–84
get() method, 12
GET requests, 83–84
getArray() function, 12
getDate() method, 160
getElementById() function, 33

ghost option, 132
Graphical effects, 191–192
greedy option, 113
grep() utility, 13–14
grid option, 107
Groups
  check boxes, 271–273
  content, 135–141
  form elements in lists, 288–289
  radio buttons, 269–271
  themes for, 257–258

H

<h3> elements, accordions from, 135–136
handle option for draggable elements, 107
handles option for resizable elements, 132
has() function, 31, 33–34
hash option, 245
header option, 139
Headers
  accordions, 139
  displaying pages with, 203–205
  fullscreen view, 231–232
  between list items, 281–283
  passing, 93–94
height option, 163
helper() function, 303
Helper functions in jQuery Mobile, 242–246
helper option for draggable elements, 107–108
Hidden text, reading, 16
hide() method, 160
hide option, 163
hideIfNoPrevNext option, 157
Hiding
  collapsible elements, 257–258
  datepicker, 160
  dialogs, 163
  elements using fade and slide, 189–191
  footers, 229–231
  HTML content, 42–43
Highlighting words, 36–37
.hmtl() function, 5
Horizontal check box groups, 271–273
Horizontal radio button groups, 269–271
host option, 245
hostname option, 245
hover() function, 35
Hover selector emulation, 34–35
hoverClass option, 113
href option, 245
hrefNoHash option, 246
hrefNoSearch option, 246
HTML
  changing between animations, 200–202
  displaying pages with headers, 203–205
  dynamic, 51–53, 85
  in elements, 46–47
  hiding content, 42–43
  loading directly, 85
  title elements, 206–208
html() function, 37
html() method, 46–47
HTML5 data- attributes, 15–16
HTTP
  listening for requests, 82–83
  passing headers, 93–94

I

Icons
  accordion, 139
  displaying, 196–198
  navigation bars, 255–256
icons option, 139

id element, 5
idPrefix option, 178
iframeFix option, 108
Image gallery carousels, 344–346
Implicit redirects, 90–91
inArray() function, 21
Indexes for array element position, 12–13
Input
  alternative keyboards, 263–265
  calendar-based, 261–263
  integers, 267–268
  numbers, 170–174
  specialized fields, 265–266
  suggesting using autocomplete, 142
  text form fields, 260–261
  validating on server side, 94–96
<input> element for keyboard events, 69–70
insertAfter() function, 58–59
insertBefore() function, 58–59
Inserting elements at specific positions, 56–58
Installing jQuery UI, 103–104
Integer input, 267–268
Interface. *See* **jQuery UI**
Internals, promises for, 311–314
Invoking callback functions, 306–308, 316–317
is() function, 31
isAbsoluteUrl() function, 246
isArray() function, 21
isDisabled() method, 160
isEmptyObject() function, 21
isFunction() function, 21
isOpen() method, dialogs, 167
isPlainObject() function, 21
isRTL option, 157
isSameDomain() function, 246
items option for sortables, 118
Iterating arrays, 8–9

J

JavaScript calls
  changing pages using, 236–239
  within HTML, 4
  loading pages using, 239–240
JavaScript Object Notation (JSON) format
  data storage, 19
  value fetching, 83–84
jqmData() function, 203, 241–242
jQuery Core
  animations, 186–187
  appending items, 49–50
  arrays. *See* Array elements
  basic usage, 3–5
  browser testing, 25–26
  callback objects, 306–308
  classes, 41–45
  content, 48–51
  elements. *See* Elements
  events. *See* Events
  functions and plugins chaining, 296–297
  HTML, 46–47, 51–53
  HTML5 attributes, 15–16
  libraries, 6–7, 103
  object extensions, 22–24
  properties and attributes, 45–46
  selections. *See* Selections
  serializing form data, 24–25
  servers. *See* Servers
  variables, 19–22
  versions, 7–8
jQuery Mobile
  attaching data to DOM nodes, 241–242
  back buttons, 221–223
  basics, 203–205
  calendar-based input, 261–263

check boxes, 271–273
collapsible elements, 257–258
collapsible sets, 258–260
color schemes, 232–234
drop-down lists, 273–276
events, 217–221
flip switches, 268–269
footers. *See* Footers
form fields for text, 260–261
headers, 231–232
helper functions, 242–246
input fields, 263–266
JavaScript calls, 236–239
linking to external pages, 212–213
lists. *See* Lists
multiple columns, 234–236
native forms without, 276–277
navigation bars, 253–257
page loading, 208–212, 239–240
radio buttons, 269–271
serving multiple pages, 205–206
sliders, 267–268
title elements, 206–208
toolbars inside content, 249–250
transition animations, 213–216
warning bars, 250–252
JQuery objects
context-aware plugins, 295–296
static plugins, 293–294
jquery property, 7
jQuery UI, 103
animating colors using, 187–189
CSS classes, 192–194
downloading and installing, 103–104
easing functions, 190, 194–196
elements. *See* Elements
graphical effects, 191–192
icon display, 196–198
styling, 185–186
widgets. *See* Widgets
JSON (JavaScript Object Notation) format
data storage, 19
value fetching, 83–84
JSONP, 98–100

K

Keyboards
alternative, 263–265
events, 69–70
keydown events, 69
keypressed events, 69
keyup events, 69

L

length() function for element sets, 14–15
length() method for tabs, 181
Libraries, 6–7, 103
Linking to external pages
with Ajax, 208–210
without AJAX, 212–213
Listeners, central event, 73–75
Listening
to form events, 70–72
for HTTP requests, 82–83
to keyboard events, 69–70
to <li> item events, 75–78
to mobile events, 217–221
to mouse events, 65–67
to scroll events, 72–73
Lists
building, 52–53
connecting, 123–124
count items with, 283–284
displaying, 278–279
filtering elements, 286–287

Lists (*contd.*)
- grouping form elements, 288–289
- headers between items, 281–283
- moving items, 58–59
- nesting, 279–280
- numbering values in, 8–9
- selecting elements, 124–130, 148–149
- sortable, 116–123
- splitting into columns, 280–281
- styling elements as buttons, 148–149
- text formatting, 285–286
- thumbnails in, 284–285
- transforming into tab panes, 175–176

live() function, 73–75, 77

load event for tabs, 179

load() function, 85

load() method for tabs, 181

Loading
- HTML directly, 85
- pages using AJAX, 208–210
- pages using JavaScript calls, 239–240
- XML, 96–97

loadPage() method, 240

lock() function, 307–308

Look and feel. *See* **jQuery UI**

Lorem Ipsum text, 142–143

M

makeArray() function, 12

makePathAbsolute() function, 246

makeUrlAbsolute() function, 246

map() function, 37

maps() function, 9–10

maxDate option, 157

maxHeight option
- dialogs, 163
- resizable elements, 133

maxWidth option
- dialogs, 163
- resizable elements, 133

Menu bars for footer elements, 252–253

Menus, drop-down, 326–328

Methods
- autocomplete, 147
- buttons, 150
- calling based on function parameters, 301–303
- datepicker, 160
- dialogs, 167
- droppable areas, 115
- plugins, 300–303
- progressbars, 170
- resizable elements, 134
- selectables, 128
- sliders, 174
- sortable, 122–123
- tabs, 180–181

min option, 172

minDate option, 157

minHeight option
- dialogs, 163
- resizable elements, 133

minLength() method, 147

minLength option, 145

minWidth option
- dialogs, 163
- resizable elements, 133

Mobile. *See* **jQuery Mobile**

modal option, 163

Modal windows, 323–326

Months, selecting, 13–14

Mouse events
- for drawing, 67–69
- listening to, 65–67

mousedown() event, 67

mouseenter() event, 67

mouseleave() event, 67

mousemove() event, 67

mouseout() event, 67

mouseover() event, 67

mouseup() event, 67

moveToTop() method, 167

Moving
- DOM tree elements, 53–54
- list items, 58–59

Multiple columns, 234–236

Multiple pages, serving, 205–206

Multiple rows in navigation bars, 254–255

Multiple themes in single pages, 232–233

N

Names
- classes, 43–45
- icon lists, 196–198

Native forms, displaying, 276–277

Navigation bars
- adding, 253–257
- icons in, 255–256
- multiple rows in, 254–255

navigation option, 139

Nested unordered lists, selectable, 129–130

Nesting
- lists, 279–280
- selectors, 33–34

Newly created items, responding to, 74–75

nextText option, 156

node command, 81

Node.js, server setup in, 81–83

not() selector, 33–34

notify() function, 316–317

notifyWith() function, 319–320

Numbering list values, 8–9

numberOfMonths option, 157

Numbers, retrieving, 170–174

O

Objects
- callback, 306–308
- deferred, 87, 199, 309–311
- extending, 22–24
- testing, 7–8

Odd elements, separating, 28–29

off() function, 77

offset() function, 69

on() function, 77–78

onChangeMonthYear event, 159

onclick attribute, 5

onClose event, 159

onmousemove event, 68

onSelect event, 159

opacity option, 108

opacity property, 26

open event
- autocomplete, 146
- dialogs, 165

open() method for dialogs, 167

option() method
- accordions, 141
- dialogs, 167
- dragging elements, 111

Ordering list items, 58–59

Orientation
- changes, 217–218
- sliders, 172

orientationchange event, 218

out event
droppable areas, 115
sortable, 121
over event
droppable areas, 115
sortable, 121

P

Page-not-found results, 89–90
pagebeforechange event, 220
pagebeforecreate event, 220
pagebeforehide event, 220
pagebeforeload event, 219
pagebeforeshow event, 220
pagechange event, 220
pagechangefailed event, 220
pageContainer option, 238
pagecreate event, 220
pagehide event, 218, 220
pageinit event, 221
pageload event, 219
pageloadfailed event, 219
pageremove event, 221
Pages
appending outcome of functions, 49–50
classes, 41–45
content, 48–51
dynamic HTML generation, 51–53
elements. *See* Elements
with headers, 203–205
HTML in elements, 46–47
loading messages, 210–212
loading using JavaScript calls, 239–240
moving list items, 58–59
properties and attributes, 45–46
redirects, 90–91
serving from single HTML file, 205–206
tabs, 174–181
pageshow event, 218, 220
panelTemplate option, 178
Paragraphs
HTML replacement in, 47
indexes, 12–13
selecting, 27–28
transforming into dialogs, 160–167
wrapping and unwrapping, 61–64
Parameterizing plugins, 297–299
Parameters
method calls based on, 301–303
plugins, 299–300
parseUrl() function, 243–246
Passing
context to callback functions, 308–309
functions, 44–45
HTTP headers, 93–94
password option, 246
Passwords input fields, 265–266
pathname option, 245
Pixels, drawing, 67–69
placeholder attribute for text fields, 261
placeholder option for sortables, 118
Plugins
callback objects, 306–308
chaining, 296–297
context-aware, 295–296
context to callback functions, 308–309, 317–319
context to progress functions, 319–320
default parameters, 299–300
deferred objects, 309–311
methods, 300–303
parameterizing, 297–299
plugins created by, 303–306

promises, 311–315
static, 293–294
third-party. *See* Third-party plugins
visualizing progress, 315–317

pop animation, 213
Popovers, 334–336
port option, 245
Position
array elements, 12–13
rectangles, animating, 186–187
position() function, 69
position option
autocomplete, 145
dialogs, 163
prepend() function, 50–51
prependTo() function, 50–51
prevText option, 156
progress() function, 316–317
Progress functions, context to, 319–320
Progressbars, 168–170
promise() function, 314
promise() method, 312
Promises
code execution at end of animations with, 198–200
for flow structures, 314–315
operation of, 86–87
to protect internals, 311–314
prop() function, 45–46
Properties
vs. attributes, 45–46
testing, 25–26
Protecting internals, promises for, 311–314
protocol option, 245
proxy() function, 78–80
Purpose-specific keyboards, 264–265

Q

querySelectorAll() function, 32–33
Question and answer pages, 341–343
Queues, 200–202

R

Radio buttons
displaying, 269–271
selecting, 32–33
range option, 172
Ranges of sliders, 172, 268
Reading
hidden text, 16
JSONP, 98–100
URLs, 243–245
XML values, 96–97
ready event, 5
receive events, 120
Rectangles, animating, 186–187
Recursive execution of effects, 191–192
Redirects, page, 90–91
Refining selections, 28–29
refresh() method, 160
refreshPositions() method for sortable, 123
refreshPositions option for draggable elements, 108
Registering callback functions, 306–308
reject() function, 315
rejectWith() function, 317–319
rel attribute, 336
reloadPage option, 238
remove event
sortable, 121
tabs, 180
remove() function for elements, 60–61
remove() method for tabs, 180

removeClass() function, 42–43
removeData() function, 18–19
Removing
classes, 42–43
CSS classes using jQuery UI, 192–194
data attached to DOM nodes, 241–242
elements, 18–19, 60–61
Repetitive element emulation, 55–56
Requests
GET, 83–84
listening for, 82–83
timeouts, 91–92
resizable component for easing functions, 194
resizable option for dialogs, 164
resize event
dialogs, 166
resizable elements, 133
resize() method for accordions, 141
resizeStart event, 166
resizeStop event, 166
Resizing
accordions, 141
dialogs, 164, 166
elements, 130–134
resolve() function, 314–315, 317–318
resolveWith() function, 317, 319
Retrieving
data attached to DOM nodes, 241–242
HTML5 data- attributes, 15–16
numbers, 170–174
Returning
deferred objects, 309–311
promises, 311–314
reverse option, 238
revert option
draggable elements, 108
sortables, 118
revertDuration option, 108
role option, 238
rotate() method, 181
Rows in navigation bars, 254–255

S

Schemes, color, 232–234
scope option
draggable elements, 108
droppable areas, 113–114
Scoping selection context, 39–40
Screen, drawing easing functions on, 195–196
script element, 5
Scroll events, 72–73
scroll option for draggable elements, 109
Scrolling text with fixed footers, 227–229
scrollSensitivity option, 109
scrollSpeed option, 109
ScrollSpy plugin, 328–331
scrollstart event, 218
scrollstop event, 218
Search and search boxes
input fields, 265–266
list filtering, 287
Twitter posts, 99–100
search event for autocomplete, 145
search option, 245
Second pages, navigating to, 205–206
with animation, 213–214
back buttons, 222–223
select event
autocomplete, 146
tabs, 179

select() method, 181
Selectable elements
events, 126–128
making, 124–125
methods, 128
nested unordered lists, 129–130
options, 126
styling, 126
selected event, 127
selected option, 178
selecting event, 127
Selections
context, scoping, 39–40
dates, 152–160
from drop-down Lists, 273–276
elements. *See* Element selection
paragraphs with spans, 27–28
Selectors
creating, 37–39
CSS hover, 34–35
nesting, 33–34
selectOtherMonths option, 157
serialize() method, 25, 122
serializeArray() function, 25
Serializing
form data, 24–25, 94–96
sortable elements, 122
Servers, 81
AJAX events, 97–98
errors, 87–88
GET requests, 83–84
loading HTML, 85
loading XML, 96–97
multiple pages, 205–206
page-not-found results, 89–90
page redirects, 90–91
passing HTTP headers, 93–94
promises, 86–87
reading JSONP, 98–100
request timeouts, 91–92
setting up, 81–83
validating form input, 94–96
setDate() method, 160
setInterval() function, 169
setTimeout() function, 312
shortYearCutoff option, 157
show event for tabs, 179
show() method for datepicker, 160
show option for dialogs, 164
showAnim option, 157–158
showButtonPanel option, 156
showCurrentAtPos option, 157
Showing
collapsible elements, 257–258
footers, 229–231
showLoadMsg option, 238
showMonthAfterYear option, 158
showWeek option, 158
Side effects of extending objects, 22–24
Single elements, selecting, 269–271
Single HTML file, serving multiple pages from, 205–206
Size
animating, 186–187
element sets, 14–15
sleep function, 91–92
Slide
animation, 213
hiding elements using, 189–191
slide event, 173–174
slidedown animation, 213
slidefade animation, 213

Sliders
for integers, 267–268
for retrieving numbers, 170–174
transforming <div> elements into, 171
slideup animation, 213
snap option, 109
snapMode option, 109
snapTolerance option, 109
sort events, 119
Sortable elements, 116–123
Sorting tree structure elements, 123–124
source option for autocomplete, 144
Spans, selecting, 27–28
Specialized input fields, 265–266
Specific positions, inserting elements at, 56–58
spinner option, 178
split() function, 37
Splitting lists into columns, 280–281
stack option
dialogs, 164
draggable elements, 109
start event
dragging elements, 110
resizable elements, 133
selectable elements, 127
sliders, 173
sortable, 119
Stateful buttons, 338–340
Static plugins, 293–294
Status of process, 168–170
step option for sliders, 173
stepMonths option, 158
stop event
dragging elements, 110
resizable elements, 134
selectable elements, 128
sliders, 174
sortable, 120
stop() function, 199
Storing
element data, 16–18
objects, 19
style attribute, 105
Styling
accordions, 136–137
autocomplete, 143
buttons, 149
datepicker, 152–154
dialogs, 161–162
draggable elements, 105
droppable areas, 113
jQuery UI components, 185–186
list of elements, 148–149
progressbars, 169
radio buttons, 151
resizable elements, 131
selectable elements, 126
sliders, 172–173
sortables, 117
tabs, 176–177
toggling, 338–340
Submit buttons, 32–33
Success, callbacks calling on, 309–311
Suggesting input values using autocomplete, 142
swipe event, 218
swipeleft event, 218
swiperight event, 218
Swipes, catching, 217–218
Switching
CSS classes, 193–194
pages, 214–216

T

tab-pane class, 332

Tabs

navigating pages using, 174–181

toggling, 331–332

transforming unordered lists into, 175–176

tabTemplate option, 178

tap event, 218

taphold event, 218

Testing

browsers for feature support, 25–26

data types, 21

objects, 7–8

variables, 19–22

Text

buttons, 149

dates, 155

filler, 142–143

formatting list items, 285–286

scrolling, 227–229

selecting, 32–33, 35–36

Text input

autocomplete, 347–348

form fields, 260–261

text option, 149

ThemeRoller tool, 185–186

Themes

changing color schemes with, 232–234

for groups, 257–258

theme roller, 176, 185–186

then() function, 314–315

Third-party plugins, 323

alerts, 336–338

button controls, 338–340

carousels, 344–346

collapsible content, 341–343

drop-down menus, 326–328

modal window display, 323–326

popovers, 334–336

ScrollSpy, 328–331

toggling tabs, 331–332

tooltips, 333–334

typeahead for autocomplete, 347–348

this keyword for plugins, 296–297

Thumbnails in list items, 284–285

timeout option, 177

Timeouts

requests, 91–92

tabs, 177

Title elements, 206–208

title option for dialogs, 164

Toggling

CSS classes, 193–194

styles, 338–340

tabs, 331–332

tolerance option

droppable areas, 114

selectable elements, 126

sortables, 118

Toolbars

fixed, 227–229

inline in content, 249–250

warning bar, 250–252

tooltip() function, 334

Tooltips, 333–334

Transforming

<div> elements into progressbars, 168–169

<div> elements into sliders, 171

elements into buttons, 147–150

<h3> and <div> elements into accordions, 135–136

Transforming (*contd.*)
paragraphs into dialogs, 160–167
unordered lists into sortables, 116–117
unordered lists into tab panes, 175–176
Transition animations, 213–216, 238
transition option, 238
Tree structures
attaching data to nodes, 241–242
moving elements around, 53–54
selecting elements, 128–130
sorting elements, 123–124
Turning off jQuery Mobile, 276–277
Twitter Bootstrap framework, 323
Twitter connections, 99–100
type() function, 21
type option, 238
Typeahead for autocomplete, 347–348
typeof() function, 21

U

ui-bar class, 250
ui-bar-b class, 250
ui-block-a class, 236
ui-block-b class, 236
ui-block-c class, 236
ui-block-d class, 236
ui-button-disabled class, 149
ui-corner class, 186
ui-draggable class, 106
ui-draggable-dragging class, 106
ui-droppable class, 113
ui-grid-a class, 236
ui-grid-b class, 236
ui-grid-c class, 236
ui-helper class, 186
ui-hidden-accessible class, 261
ui-li-aside class, 286
ui-li-count class, 284
ui-li-icon class, 285
ui object, 118
ui-priority class, 186
ui-resizable class, 131, 186
ui-selectable class, 126
ui-selected class, 126
ui-selectee class, 126
ui-selecting class, 126
ui-sortable class, 117
ui-sortable-helper class, 117
ui-state class, 186
ui-state-active class, 149
ui-state-disabled class, 149
ui-state-highlight class, 154
ui-state-hover class, 149
ui-unselecting class, 126
ui-widget class, 186
<ul> items, 75–78
unbind() function, 67
undelegate() function, 75
Unordered lists
building, 52–53
connecting, 123–124
selecting elements in, 124–130
sortable, 116–123
transforming into tab panes, 175–176
unselected event, 128
unwrap() function, 63–64
Unwrapping elements, 61–64
Up buttons, 58–59
update event, 120
updatelayout event, 221
url() method, 181
URLs
loading, 181
reading, 243–245

User interface. *See* **jQuery UI**
username option, 246
Users, alerts for, 336–338

V

Validating form input, 94–96
value option
progressbars, 169
sliders, 173
Variables, testing and manipulating, 19–22
Variants, drop-down, 274–275
vclick event, 219, 221
Versions
determining, 7–8
upgrading, 5
Vertical check box groups, 271–273
Vertical radio button groups, 269–271
Visualizing progress before callbacks, 315–317
vmousecancel event, 219
vmousedown event, 219
vmousemove event, 219
vmouseover event, 218
vmouseup event, 219

W

Warning bars, 250–252
-webkit- prefix, 216
weekHeader option, 158
widget() method
dialogs, 167
dragging elements, 111
Widgets, 135
accordions, 135–141
autocomplete, 142–147
buttons, 147–150
buttonsets, 151
datepicker, 152–160
dialogs, 160–167
progressbar, 168–170
sliders, 170–174
tabs, 174–181
width option, 163
Words, highlighting, 36–37
wrap() function, 61–62
wrapAll() function, 63
wrapInner() function, 62–63
Wrapping elements, 61–64

X

XML, loading, 96–97
XmlHttpRequest (XHR) object, 98

Y

yearRange option, 158
yearSuffix option, 158
Yellow toolbar, 250–252

Z

zIndex option
dialogs, 164
draggable elements, 109