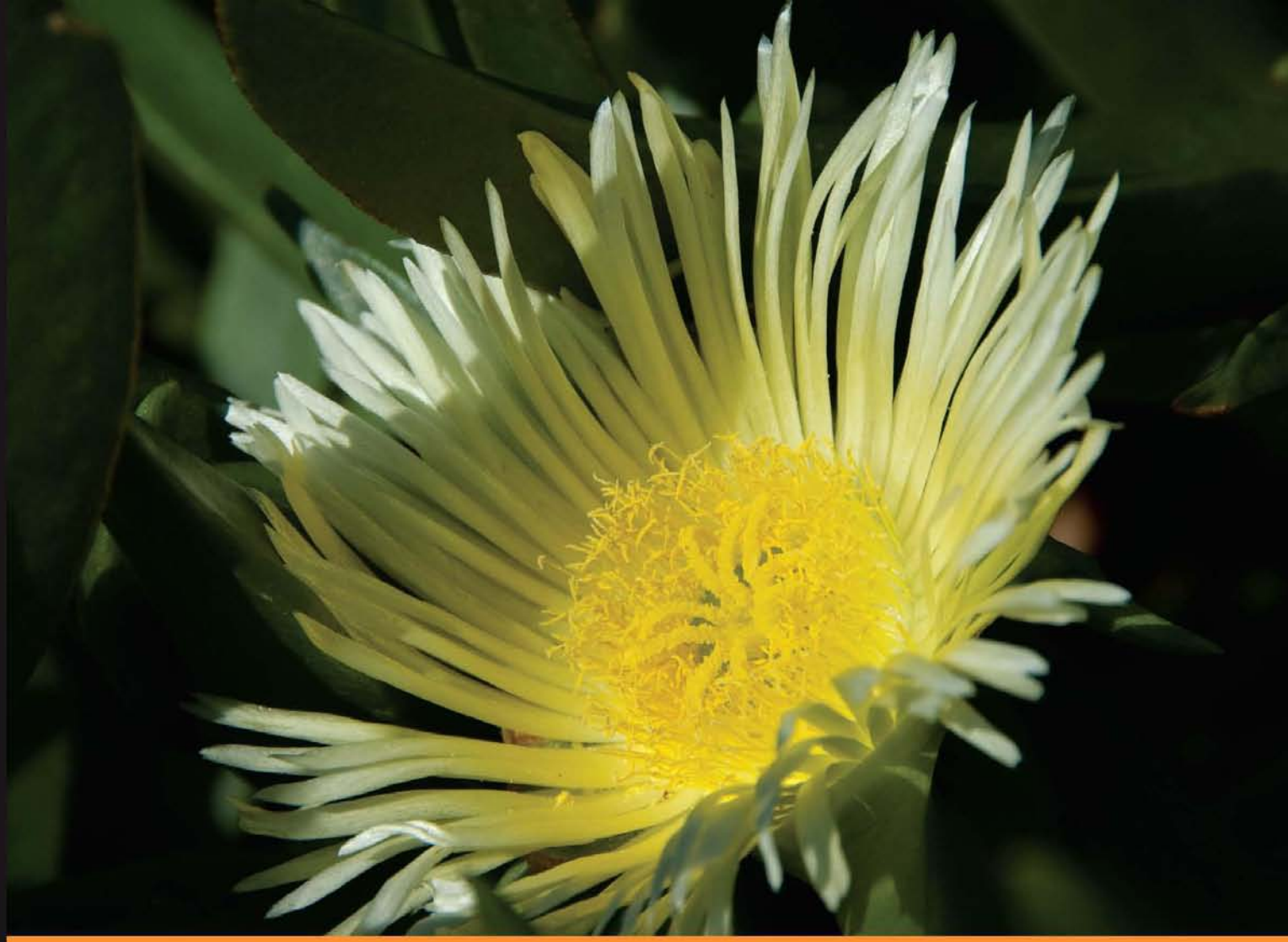


From Technologies to Solutions

# jQuery UI 1.6

## The User Interface Library for jQuery

Build highly interactive web applications with ready-to-use widgets from the jQuery user interface library

**Dan Wellman**

PACKT PUBLISHING

# jQuery UI 1.6

## The User Interface Library for jQuery

Build highly interactive web applications with ready-to-use widgets from the jQuery user interface library

**Dan Wellman**

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

# jQuery UI 1.6

## The User Interface Library for jQuery

Copyright © 2009 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, Packt Publishing, nor its dealers or distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: January 2009

Production Reference: 1200109

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847195-12-8

www.packtpub.com

Cover Image by Karl Swedberg (karl@englishrules.com)

# Credits

**Author**

Dan Wellman

**Reviewers**

Akash Mehta

David Methvin

Mark Grabanski

**Senior Acquisition Editor**

Douglas Paterson

**Development Editor**

Ved Prakash Jha

**Technical Editor**

Dhiraj Bellani

**Editorial Team Leader**

Akshara Aware

**Production Editorial Manager**

Abhijeet Deobhakta

**Projects Team Leader**

Lata Basantani

**Project Coordinators**

Neelkanth Mehta

Brinell Lewis

**Indexer**

Hemangini Bari

**Proofreaders**

Joel Johnson

Camille Guy

**Production Coordinator**

Aparna Bhagat

**Cover Work**

Aparna Bhagat

# About the Author

**Dan Wellman** lives with his wife and children in his home town of Southampton on the south coast of England. By day, his mild-mannered alter-ego works for a small, yet accomplished, e-commerce production agency. By night, he battles the forces of darkness and fights for truth, justice, and less intrusive JavaScript.

Dan has been writing computer-related articles, tutorials, and reviews for around five years and is rarely very far from a keyboard of some description.

This is his second book.

---

I'd like to say a big, personal thank you to the Packt editorial team for their continued support and encouragement, and to the jQuery UI development team for producing such an incredible library. Special thanks also goes out to Eamon O'Donoghue for his invaluable graphic/imagery advice, and to Mike Newth for his equally invaluable printing assistance. Finally, I'd like to thank James Zabiela, Andrew Herman, Steve Bishop, Aaron Matheson, Dan Goodall, Mike Woodford, and John Adams; they contributed in no way towards this book, but are nevertheless the greatest bunch of dudes a guy could hang out with.

---

# About the Reviewers

**Akash Mehta** is a web application developer, technical writer, and business consultant based in Brisbane, Australia. His past projects include brochure websites, e-learning solutions, and information systems. He has written web development articles for several publishers in print and online. He is a regular speaker at local conferences, and contributes to prominent PHP blogs.

As a student, Akash maintained PHP web applications and built user interfaces using the jQuery toolkit. While pursuing both a commerce and an IT degree, Akash develops web applications on PHP and Python platforms. After hours, he organizes his local PHP user group.

Akash develops applications on a wide range of open source libraries. His toolbox includes a number of application frameworks, including the Zend Framework, CakePHP, and Django; Javascript frameworks, including jQuery, Prototype and Mootools; platforms such as Adobe Flash/Flex; and the MySQL and SQLite database engines.

Currently, Akash provides freelance technical writing and web development through his website, `http://bitmeta.org`.

**David Methvin** is the Chief Technology Officer at PC Pitstop, and one of the founding partners of the company. He provides technical direction for the `www.pcpitstop.com` website, oversees software development, and serves as an editor for the site's content. David also serves as the site's chief investigator for new spyware and adware threats. Before joining PC Pitstop, David had an extensive career in computer journalism. He served as an Executive Editor at both Windows Magazine and PC Tech Journal, co-authored a book on Windows NT networking, and wrote articles for more than two dozen publications. David continues to write a monthly column, "The Well-Tuned PC," for `pctoday.com` magazine. David holds both Bachelor's and Master's Degrees in Computer Science from the University of Virginia. After graduation, he spent several years designing and developing software for robotics and telecommunications systems with companies such as General Electric.

**Marc Grabanski** runs a consulting company focused on user interface development and user experience. He believes firmly that open source is the fuel that helps a developer go from mediocre to excellent at what they do. His work with jQuery was no exception. By writing jQuery UI Datepicker, he met so many great people and is thankful for the experiences gained by doing this project.

---

I want to thank, of course, John Resig for writing jQuery and the whole jQuery UI team for carrying the library to new levels each day.

---

*This book is dedicated to my supportive and understanding wife Tammy.*
*Thanks babe.*

# Table of Contents


**Preface** 1

**Chapter 1: Introducing jQuery UI** 9

**Is this book for me?** 10
**Downloading the library** 11
**Setting up a development environment** 12
**The structure of the library** 13
Unit testing 14
Widget theming 15
Minified and packed components 15
**Theme Roller** 16
**The simplified API** 17
**Component categories** 18
**Browser support** 19
**Book examples** 20
**Library licensing** 21
**Summary** 21

**Chapter 2: Tabs** 23

**A basic tab implementation** 24
**Tab styling** 26
**Configurable properties** 28
**Transition effects** 31
**Tab events** 33
**Using tab methods** 37
Enabling and disabling tabs 37
Adding and removing tabs 39
Simulating clicks 42
Creating a tab carousel 43
**AJAX tabs** 46

**Fun with tabs** **52**
**Summary** **56**
**Chapter 3: The Accordion Widget** **57**
**Accordion's structure** **58**
**Styling the accordion** **61**
**Configuring accordion** **65**
**Accordion methodology** **72**
Destruction 72
Enabling and disabling 74
Drawer activation 78
**Accordion animation** **79**
**Accordion events** **81**
**Fun with accordion** **83**
**Summary** **87**
**Chapter 4: The Dialog** **89**
**A basic dialog** **90**
**Custom dialog skins** **92**
**Dialog properties** **94**
Adding buttons 98
**Working with dialog's callbacks** **100**
**Using dialog animations** **102**
**Controlling a dialog programmatically** **104**
**Getting data from the dialog** **108**
**Fun with dialog** **111**
**Summary** **116**
**Chapter 5: Slider** **117**
**Implementing slider** **118**
**Overriding the default theme** **119**
**Configurable properties** **122**
**Using slider's callback functions** **125**
**Slider methods** **127**
**Slider animation** **131**
**Multiple handles** **131**
**Fun with slider** **134**
**Summary** **138**
**Chapter 6: Date Picker** **139**
**The default date picker** **140**
**Skinning the date picker** **142**
**Configurable properties of the picker** **144**
Changing the date format 148

Localization 150
Callback properties 156
Trigger buttons 159
Multiple months 161
Enabling range selection 163
Configuring alternative animations 164
Date picking methods 166
Putting the date picker in a dialog 168
**Fun with date picker 175**
**Summary 182**
**Chapter 7: Auto-Complete 183**
**Basic implementation 184**
**Configurable properties 186**
Scrolling 190
Auto-complete styling 192
Multiple selections 197
Advanced formatting 198
Matching properties 205
Remote data 207
Sending additional data to the server 210
Caching 210
**Auto-complete methods 211**
**Fun with auto-complete 214**
**Summary 218**
**Chapter 8: Drag and Drop 219**
**The deal with drag and droppables 220**
**Draggables 221**
A basic drag implementation 221
**Configuring draggable properties 223**
Resetting dragged elements 227
Drag handles 228
Helper elements 230
Constraining the drag 233
Snapping 236
**Draggable event callbacks 238**
**Using draggable's methods 243**
**Droppables 244**
**Configuring droppables 247**
Tolerance 251
**Droppable event callbacks 254**
Greed 257

**Droppable methods** **261**
**Fun with droppables** **261**
**Summary** **267**
**Chapter 9: Resizing** **269**
**A basic resizable** **270**
**Skinning the resizable** **273**
**Resizable properties** **274**
Configuring resize handles 275
Defining size limits 279
Resize ghosts 280
Constraining the resize and maintaining ratio 282
Resizable animations 284
**Resizable callbacks** **286**
**Resizable methods** **289**
**Fun with resizable** **289**
**Summary** **292**
**Chapter 10: Selecting** **293**
**Basic implementation** **294**
**Selectee class names** **297**
Configurable properties of the selectable class 298
Filtering selectables 299
**Selectable callbacks** **301**
**Selectable methods** **304**
**Fun with selectables** **308**
**Summary** **320**
**Chapter 11: Sorting** **321**
**Basic implementation** **321**
**Configuring sortable properties** **325**
Placeholders 331
Sortable helpers 334
Sortable items 336
Connected lists 338
**Reacting to sortable events** **341**
**Connected callbacks** **347**
**Sortable methods** **351**
**Widget compatibility** **354**
**Fun with sortable** **356**
The main script 360
**Summary** **373**

**Chapter 12: UI Effects** **375**
**The core effects file** **376**
Color animations 376
Class transitions 378
Advanced easing 380
**Highlighting** **381**
Additional effect parameters 382
**Bouncing** **384**
**Shaking** **385**
**Transference** **387**
**Scaling** **390**
**Element explosion** **392**
**The puff effect** **395**
**Pulsate** **397**
**Drop** **399**
**Slide** **402**
**Clip** **405**
**Blind** **407**
**Fold** **409**
**Summary** **411**
**Index** **413**

# Preface

jQuery has been a phenomenal success, with many newcomers to the world of JavaScript frameworks choosing it, and many developers moving to it from other frameworks. jQuery UI is tipped to follow suit, and has already seen massive growth and take-up, with more success to follow. By learning how to use it now, you can be a part of its success.

jQuery UI is a collection of engaging widgets and essential interaction-helpers that can help drastically reduce the amount of code you need to write and the amount of time you need to spend developing. Each component comes with a wide range of easy-to-use configurable properties and methods. The components all share a common programming interface that quickly becomes second nature to work with.

Development of the library is ongoing, with highly-skilled developers building it to ever greater levels. New components are being added between major releases, and bug fixes/updates are constantly being tested and applied. jQuery UI is very much a community-driven site, with the team relying on and building upon bug reports and feature requests submitted by the wider development community. Additionally, new components are often derived from the most useful jQuery plug-ins.

This book will give you a head start in learning jQuery UI; exposing the APIs behind the complete range of components as of version 1.5.4, and including much of the functionality of version 1.6 of the library. Each component is methodically and consistently looked at, with many functional examples. Each chapter ends with a full implementational example, looking at ways in which the components can be used in real-world scenarios.

# What this book covers

*Chapter 1* A general overview of jQuery UI. You'll find out exactly what the library is, where it can be downloaded from and where resources for it can be found. You'll look at the freedom the license gives you to use the library, and how the API has been simplified to give the components a consistent and easy-to-use programming model.

*Chapter 2* We begin our journey through jQuery UI by looking at the high-level user interface widgets, focusing on the tabs component; a simple but effective means of presenting structured content in an engaging and interactive widget.

*Chapter 3* Next, we take a look at the accordion widget. This is another component dedicated to the effective display of content. Highly engaging and interactive, the accordion makes a valued addition to any web page and its API is exposed in full to show exactly how it can be used.

*Chapter 4* In this chapter, we focus on the dialog widget. The dialog behaves in the same way as a standard browser alert, but it does so in a much less intrusive and visitor-friendly manner. We look at how it can be configured and controlled to provide maximum benefit and appeal.

*Chapter 5* The slider widget provides a less commonly used, but no less valued, user interface tool for collecting input from your visitors. We look closely at its API throughout this chapter to see the variety of ways in which it can be implemented.

*Chapter 6* Next, we look at the date picker. This component packs a huge amount of functionality and appeal into an attractive and highly usable tool allowing your visitors to effortlessly select dates. We look at the wide range of configurations that its API makes possible as well as seeing how easy common tasks, such as skinning and localization, are made.

*Chapter 7* The last widget we look at is the auto-complete; a highly professional and desired addition to any page. We'll look at the different data sources you can provide and how to customize the widget, in addition to seeing which properties and methods we have at our disposal. At the time of writing, the latest stable version of the UI library is 1.5.4, but this widget is part of 1.6 release, a sneak preview of what we've got to look forward to.

*Chapter 8* We begin looking at the low-level interaction helpers in this chapter, tackling first the related draggable and droppable components. We look at how they can be implemented individually and how they can be used together for maximum effect.

*Chapter 9* In this chapter, we look at resizing component and see how it is used with the dialog widget. We see how it can be applied to any element on the page to allow it to be resized in a smooth and attractive way.

*Chapter 10* Next, we look at the selectable component, which allows us to add behavior to elements on the page and allow them to be selected individually or as a group. We see that this is one component that really brings the desktop and the browser together as application platforms.

*Chapter 11* We look at the final interaction helper in this chapter – the sortable component. This is an especially effective component that allows you to create lists on a page that can be reordered by dragging items to a new position on the list. This is another component that can really help you to add a high level of professionalism and interactivity to your site with a minimum of effort.

*Chapter 12* The last chapter of the book is dedicated solely to the special effects that are included with the library. We look at an array of different effects that allow you to show, hide, move, and jiggle elements in a variety of attractive and appealing animations. There is no 'fun with' section at the end of this chapter; the whole chapter is a 'fun with' section.

# What you need for this book

Very little is required in order to start using jQuery UI. Specifically, you will need the following environment:

- A text editor; be it Notepad, or a full-blown development application like Dreamweaver
- A browser
- A copy of jQuery and a copy of jQuery UI
- An internet connection for dynamic data retrieval in some of the examples

# Who is this book for

This book is for front-end designers and developers who need to quickly learn how to use the jQuery UI User Interface Library. To get the most out of this book, you should have a good working knowledge of HTML, CSS, and JavaScript, and will need to be comfortable using jQuery, the underlying foundation of jQuery UI.

# Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "It is easy to achieve this by manipulating the `disabled` property of the tabs."

A block of code will be set as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 3</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //define config object
       var tabOpts = {
         selected: 1
      };
      //create the tabs
      $("#myTabs").tabs(tabOpts);
```

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

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 4</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //define config object
      var tabOpts = {
         selected: 1,
         disabled: [0]
      };
       //create the tabs
       $("#myTabs").tabs(tabOpts);
     });
   </script>
  </body>
</html>
```

**New terms** and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "Another problem we have with our test page is that clicking the `Enable!` button while the accordion is already enabled does nothing."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

# Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

# Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

# Downloading the example code for the book

Visit `http://www.packtpub.com/files/code/5128_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

# Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

# Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide the location address or website name immediately so we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

# Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

# 1
# Introducing jQuery UI

Welcome to jQuery UI 1.6: The User Interface Library for jQuery. This resource aims to take you from the first steps to an advanced usage of the JavaScript library of UI widgets and interaction helpers built on top of the awesome jQuery.

jQuery UI extends the underlying jQuery library to provide a suite of rich and interactive widgets, and code-saving interaction helpers, built to enhance the user interfaces of your websites and applications.

Because jQuery UI runs on top of jQuery, the syntax used to initialize, configure, and manipulate the different components is in the same comfortable, easy-to-use, and short-hand style that we've all come to know and love through using jQuery. Therefore, getting used to it is incredibly easy.

You also automatically get all of the great jQuery functionality at your disposal when using jQuery UI. So when you implement any particular component, your code will usually be a mixture of jQuery and jQuery UI specific code, as well as some traditional JavaScript occasionally.

We won't be looking at any code in this chapter. There are just a few points that I would like to mention before we break out the text editors and get down to some coding. In this chapter, we'll be looking at the following subjects:

- Who this book is written for
- How to obtain a copy of the library
- How to set up a development environment
- The structure of the library
- Theme Roller
- The format of the API
- Browser Support
- How the library is licensed

# Is this book for me?

This book is for developers who want to quickly and easily build engaging, highly interactive interfaces for their web applications, or less commonly, for embedded applications. I mention embedded applications because jQuery UI is suitable for other mediums than just the Internet.

Nokia was the first mobile phone company to announce that they were adopting jQuery to power parts of their cell phone operating system. I'm sure that by the time this book is published there will be more companies adapting the library for their own needs, and wherever jQuery goes, jQuery UI can follow.

People that are comfortable with HTML, JavaScript, and CSS, and have at least some experience with jQuery itself will get the most benefit from what this book has to offer. However, no prior knowledge of the UI library itself is required.

Consider the following code:

```
$("#myEl").click(function() {
  $("<p>").attr("id, "new").css({
    color:"#000000"
  }).appendTo("#target");
)};
```

If you cannot immediately see, and completely understand, what this simple code does, you would probably get more from this book after first learning about jQuery itself. Consider reading Karl Swedberg and Jonathan Chaffer's excellent **Learning jQuery**, also by Packt, or visit `http://www.learningjquery.com` and then come back to this book.

Each jQuery UI specific method or property that we work with will be fully covered in the explanatory text that accompanies each example, and where it is practical, some of the standard jQuery code will also be discussed.

Basic concepts of using jQuery itself won't be covered. Therefore, you should already be familiar with advanced DOM traversal and manipulation, attribute and style getting and setting, and making and handling AJAX calls. You should be comfortable with the programming constructs exposed by jQuery such as method chaining and using callback functions.



Knowing and understanding how jQuery works is important if you want to learn how to leverage the full potential of jQuery UI. Using jQuery promotes writing code in a particular style that is easily recognizable. Code written for jQuery UI naturally assumes this same style, and you should be comfortable enough with it to be able to easily see what is going on with different bits of code in the examples.

# Downloading the library

There are several different options for downloading the library. You can choose to download a personalized package tailored to your individual needs using the download builder, download the full development bundle containing all library files including full, packed, and minified versions of each script file, or download individual files from the online SVN repository.

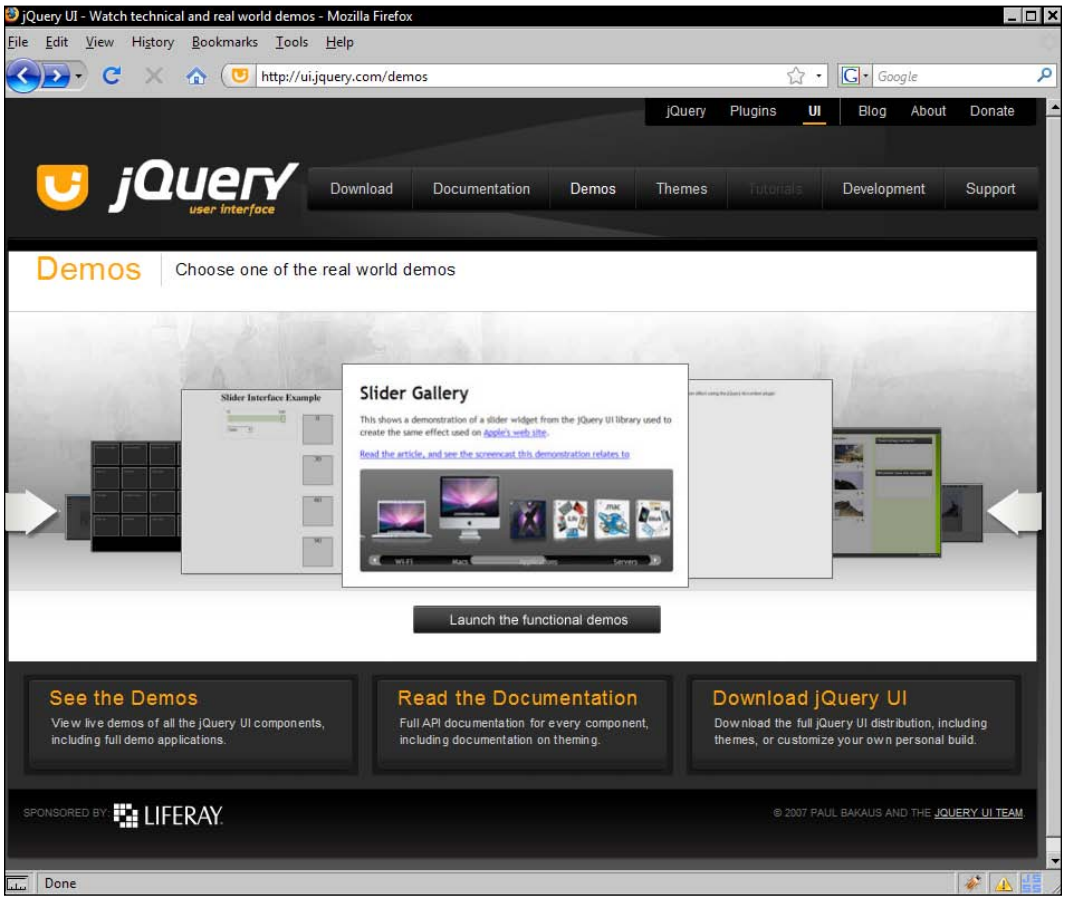
Once you've mastered jQuery UI, and are regularly using selected widgets in different projects that you're involved in, using the download builder to quickly put together the files and their dependencies that you require will be an effective way of minimizing the library's footprint within your applications.

As we'll be looking at each of the lower-level interaction components and the higher-level widgets that make up the library, we'll be working mostly with the full development bundle throughout the book. At this point, you should probably download a copy of the library, which can be obtained from the jQuery home page at `http://www.jquery.com`.

There are two versions of the full development bundle. The latest version and the most stable version. We'll be working with the latest version in our examples to make sure we get to see the newest, most cutting-edge features.

There are bugs in the code with a couple of the library components, or certain features that we want to use that aren't currently available. This is where the SVN nightlies come in, as we can link to or download the latest, most bug-free versions of each file. These are the actual working files which developers who build the library use, and have brand new fixes and patches in place.

In addition to the library itself, the jQuery UI project site is home to an extensive range of examples of different library components which are presented in a beautiful carousel-type format, with support information as seen here:




# Setting up a development environment

We'll need a location to unpack the jQuery UI library in order to easily access the different parts of it within our files. We should first create a project folder, into which all of our example files, as well as all of the library and other associated resources such as stylesheets and images, can be kept.

Create a new directory on your C: drive, or in your home directory, and call it `jqueryui`. This will be the root folder of our project and will be the location where we store all of the files that we're going to create.

To unpack the library, open it in a compression program, such as Winzip, and drag the `jqueryui1.6rc2` folder into the directory we just created. We also need to create img and styles This will give us the correct folder structure to work from. The folder structure should be as follows:

- `jqueryui`
  - `jqueryui1.6rc2`
    - `_MACOSX`
    - `demos`
    - `tests`
    - `themes`
    - `ui`
  - `img`
  - `styles`

# The structure of the library

Let's take a moment to look at the structure of the unpacked library. This will give us a feel for its composition and where the different resources that we'll be working with reside. Open up the `jqueryui1.6rc2` folder where we extracted the archived library. The contents of this folder should be as follows:

- `_MACOSX` directory
- `demos` directory
- `tests` directory
- `themes` directory
- `ui` directory
- GPL-License file
- MIT-License file
- the jQuery library
- a version file

The `_MACOSX` folder can safely be ignored, even by Mac users. It exists only because the current version of the library was opened by someone using a Mac, and I have mentioned it here solely because it exists in the unpacked structure of the library. Don't even worry about removing this folder as it's only a couple of bytes in size.

The contents of the `demos` directory shows you a series of functional, as well as real-world examples of how the different UI components can be used. Each of the components has its own demo page which is designed to work as is from its current location.

The functional example pages show a basic implementation of each component, as well as exposing some of the more common configurations that can be set. These pages are an exact mirror of their online-equivalents.

The real-world examples highlight one particular feature of a component, and demonstrate this feature by itself on the page with little or no explanatory text. While these are the same examples found on the jQuery UI project page, they are presented much better online.

# Unit testing

The `tests` folder is similar to the `demos` folder in that it contains a series of pages that highlight different features of the components found in the library. The important difference is that components which are still in beta stage, under full development, are also included. By taking a look into this folder, you can get a very good idea of what is about to be released. These examples can be found in the `visual` folder within the `tests` folder.

If you're concerned with the size of the library on your web server, or the bandwidth that uploading it would take, the `demos` and `tests` folders can safely be deleted. However, they do only take up a few megabytes of space.

Several other important resources can also be found in the `tests` folder. The `qunit` folder contains jQuery's unit testing environment. Some of you may have heard of, or used, the popular JUnit. This is a Java-based unit testing environment. QUnit is the same, but is tailored specifically for use when writing jQuery plugins or jQuery UI widgets.

For those of you who haven't done any unit testing before, this refers to the practice of writing tailored code which tests the functionality of the smallest unit available for testing within an application. In a language like JavaScript, the unit, or smallest possible abstraction of functionality, is typically a single method.

We could argue that a variable is the smallest unit within a JavaScript application, but as functions, and therefore methods, can be assigned to a variable, this wouldn't always be a viable argument.

We won't be using QUnit in any of our book examples because we won't be creating any plugins or widgets of our own. Some excellent documentation of QUnit is provided on the jQuery site for those of you who can envisage yourself doing this at some point in the future.

The `simulate` folder contains a plugin written by Eduardo Lundgren and Richard D. Worth that is used to assist unit testing with jQuery. It adds a set of methods to your toolkit that allows you to simulate common mouse and keyboard events from your code. This is useful for checking actions like drag-and-drop when this behavior is included in your widgets or plugins.

# Widget theming

The library ships with two themes. The `default` theme is light-grey and neutral looking. The `flora` theme consists of pleasant light-green and orange tones. Both provide styling for each of the higher-level widgets and can be used completely out-of-the-box without modification if desired.

Some of the CSS found in these themes go beyond mere aesthetics and instead relates to how the widget functions. Therefore, if we want to provide a custom skin for any particular widget we have two options. First, we can omit the widget's skin file completely and use our own CSS file instead of the corresponding theme file. Or second, we can simply override the rules that deal specifically with appearance.

The first method, while equally viable, creates much more work for us and essentially means we have to reinvent the wheel. By this I mean we would have to spend time writing styling code related to functionality which has already been written. The second option is much more efficient and allows us to focus on writing the barest, minimum styling code, building on the foundation already provided by the existing themes.

# Minified and packed components

The `ui` folder contains all of the un-minified versions of the code files for each component and effect, several subdirectories containing the minified and packed versions of the components, and the `i18n` folder.

The full-sized versions of each library component and effect are useful for development purposes. These can be opened up and read to get a better feel for how a particular component works. These files are complete with comments that advise us how particular sections of code work.

The minified versions of each component are excellent for production use, where downloading and interpretation of the files matters the most. JavaScript can easily be minified using a growing number of tools.

Minified files have all comments, whitespace, and line breaks removed from them. Most minification tools also obfuscate the code which shortens object, variable, and function names to just one character where possible. The code in the file is not changed in the way that it works.

The packed versions of each file are the smallest form of each component, but they are not actually minified in the above sense. Instead, these files are compressed, which is what makes them smaller than the minified files. The code within packed files is changed however, and it takes some additional client-side code to uncompress them. This means that although smaller, packed files will generally take longer to interpret.

The `i18n` directory is where the language packs for the date picker widget reside. The date picker (which we'll look at in detail in chapter 6) is very easy to internationalize using these plugin language packs.

# Theme Roller

Theme Roller is a custom tool written in jQuery that allows us to visually produce our own custom jQuery UI theme and package it up in a convenient, downloadable archive which we can drop into our project with no further coding (other than using the stylesheet in a HTML `<link>` element of course).

Theme Roller was created by Filament Group Inc and makes use of a number of jQuery plugins released into the open-source community. It can be found at `http://ui.jquery.com/themeroller`.

Theme Roller is certainly the most comprehensive tool available for creating your own jQuery UI themes. We can very quickly and easily create an entire theme comprised of all of the styles needed for targeting elements, including the images we'll need, which is compatible with all of the non-beta widgets.



The previous screenshot shows the Theme Roller interface and as you can see, it's remarkably simple to use. The top of the page, in the previous screenshot, shows a series of select boxes and input fields arranged in a tabular format.

Each column of fields represents an aspect of each widget. We can set the color and texture of the background, the border color, text color, and icon color. The icon setting refers to elements of each widget, such as the *left* or *down* icons shown on clickable areas.

Each row of settings corresponds to a state. There is the default state, the hover and active states, and the content area. The content area may be the panel of a set of tabs, or an accordion, for example. We can also set the global `font-family`, `font-style`, and the `font-size` too.

When you interact with the top half of the page, the bottom of the page, which contains a selection of example widgets, is automatically updated with your selections so you can quickly see how your theme will look.

If you're not feeling particularly inspired when creating a theme, there is also a gallery of pre-configured themes that you can instantly use. Aside from this convenience, the best thing about these preselected themes is that when you select one, it is loaded automatically into the first page of Theme Roller. Therefore, you can easily make little tweaks as you see fit.

Without a doubt, this is the best way to create a visually appealing custom theme that matches the UI widgets to your existing site. However, we won't be looking at this tool again for the remainder of this book. We'll be focusing instead on learning the style rules that we need to manually override to generate our desired skins.

# The simplified API

The version 1.5 release of jQuery UI was a milestone in the library's history. This was the release in which the API for each component was significantly simplified, making the library both easier to use and more powerful.

Once you've worked with one of the components from the library, you'll instantly feel at home when working with other components since the methods of each component are called in exactly the same way.

Methods are consistently called throughout the components by passing the method name as a simple string to the component's constructor method, with any arguments that the method accepts passed as strings after the method name. For example, to call the `destroy` method of the tabs component, we would simply do this:

```
$("#someElement").tabs("destroy");
```

See how easy that was? Every single method exposed by all of the different components is called in this same simple way. Using jQuery UI feels just like using jQuery itself and having built up confidence coding with jQuery, moving on to jQuery UI is the next logical step to take.

Many of the components also share a similar method-set of exposed functionality. For example, every single component found in the library has `destroy`, `enable`, and `disable` methods, and many others expose similar functionality. This again makes each component exceptionally easy and intuitive to use.

# Component categories

There are two types of components found within the jQuery UI library. Low-level interaction helpers that are designed to work, primarily, with mouse events, and there are the widgets, which produce visible objects on the page which are designed to perform a specific function.

The interaction-helpers category, which forms the underlying core of the library, includes the following components:

- draggable
- droppable
- resizable
- selectable
- sortable

The higher-level widgets, which often build upon the foundation provided by the lower level components, include:

- accordion
- auto complete
- date picker
- dialog
- slider
- tabs

The `ui.core.js` file, which is required by all other library components, comes under neither category, but could nevertheless be seen as a component. This file sets up the construct that all widgets use to function and adds some core functionality which is shared by all of the library components. This file isn't designed to be used on its own, and exposes no functionality that can be used outside of another component.

Apart form these components, there is also a series of UI effects, which was once a completely separate sister library called Enchant. These effects produce different animations or transitions of targeted elements on the page. These are excellent for adding flair and style to your pages, in addition to the rock-solid functionality of the components. We'll be looking at these effects in the final chapter of the book.

I'd like to add here that the jQuery UI library is currently undergoing a rapid period of expansion and development. It is also constantly growing and evolving with bug-fixes and feature enhancements continually being added. It would be impossible to keep entirely up-to-date with this aggressive expansion and cover components that are literally about to be released.

The great thing about jQuery UI's simplified API is that once you have learned to use all of the existing components, as this book will show you, you'll be able to pick up any new components very quickly. As this book is being written, there are already a number of new components nearing release, with many more in the pipeline.

Due to its success in the development community, jQuery UI is sure to become a stalwart of modern web design and is therefore worth investing time and effort in.

# Browser support

Like jQuery, jQuery UI supports all of the major browsers in use today including the following:

- IE6, IE7 and IE8
- Firefox 2 and Firefox 3
- Opera 9
- Safari 3
- Chrome

This is by no means a comprehensive list, but I think that it includes the browsers that are most likely to be used by any average web surfer. The widgets are built from semantically correct HTML generated as needed by the components. Therefore, we won't see excessive or unnecessary elements being created or used.

I'm sure I needn't remind you that your own style of coding should follow the lead of jQuery UI. You should always strive to maintain an accessible inner core of content that has successive layers of presentation and functionality layered on top in the manner of progressive enhancement.

# Book examples

The library is as flexible as standard JavaScript, and by this I mean that there is often more than one way of doing the same thing, or achieving the same end. For example, the callback properties used in the configuration objects for different components, can usually take either references to functions or inline anonymous functions, and use them with equal ease and efficiency.

In practice, it is usually advised to keep your code as minimal as possible (which jQuery really helps with anyway), but to make the examples more readable and understandable, we'll be separating as much of the code as possible into discrete modules. Therefore, callback functions and configuration objects will be defined separately from the code that calls or uses them.

To reduce the number of files that we have to create and work with, all of the JavaScript will go into the host HTML page on which it runs, as opposed to in separate files. This isn't necessary, or indeed recommended at all in fact, for production websites or applications.

I'd also just like to make it clear that the main aim throughout the course of this book is to learn how to use the different components that make up jQuery UI. If an example seems a little convoluted, it may simply be that this is the easiest way to expose the functionality of a particular method or property, as opposed to a situation that we would find ourselves coding for in a regular implementation.

Although the lower-level components provide a foundation which is built upon by the high-level widgets, we're going to be approaching the library from the opposite direction. First, we're going to look at the widgets, as these, for the most part, have smaller APIs and are therefore easier to learn and use. Once we have mastered the widgets, we're then going to peel away the outer layers to expose the inner core of functionality imparted by the interaction helpers.

At the time of writing the latest version of jQuery UI is 1.6rc2, so this is used throughout the examples. Release Candidate 3 is imminent however, and will no doubt be shortly followed by the full stable 1.6 release. So, by the time you read this, it will probably be available.

Please ensure that when working with the examples in the code download, or writing the examples yourself, you point to the correct path for the version of the library that you download.

# Library licensing

Like jQuery, the jQuery UI library is dual licensed under the MIT and GPL open-source licences. These are both very unrestrictive licenses that allow the creators of the library to take credit for its production and retain intellectual rights over it, without preventing us as developers from using the library in any way that we like.

The MIT license explicitly states that users of the software (jQuery UI in this case) are free to use, copy, merge, modify publish, distribute, sublicense, and sell. This lets us do pretty much whatever we want with the library.

The only requirement imposed by this license is that we must keep the original copyright and warranty statements intact.

This is an important point to make. You can take the library and do whatever you like with it. Build applications on top of it and then sell those applications, or give them away for free. Put the library in embedded systems like cell-phone OSs and sell those. But whatever you do, leave the original text file with John Resig's name in it present. You may also duplicate it word for word in the help files or documentation of your application.

The MIT license is very lenient, but because it is not copyrighted itself, we are free to change it. We could therefore demand that users of our software give attribution to us instead of the jQuery team, or pass off the code as our own.

The GPL license is copyrighted, and offers an additional layer of protection for the library's creators and the users of our software. Because jQuery is provided free and open-source, the GPL license ensures that it will always remain free and open-source, regardless of the environment it may end up in, and that the original creators of the library are given the credit they deserve. Again, the original GPL license file must be available within your application.

# Summary

jQuery UI removes the difficulty of building engaging and effective user interfaces. It provides a range of components that can quickly and easily be used out-of-the-box with little configuration. If a more complex configuration is required, they each expose a comprehensive set of properties and methods for integration with your pages or applications.

Each component is designed to be efficient, light-weight, and semantically correct, and makes use of the latest object-oriented features of JavaScript. When combined with jQuery, it provides an awesome addition to any web developer's toolkit.

So far, we've seen how the library can be obtained, how your system can be set up to utilize it, and how the library is structured. We've also looked at how the different widgets can be themed or customized, how the API simply and consistently exposes the library's functionality, and the different categories of component.

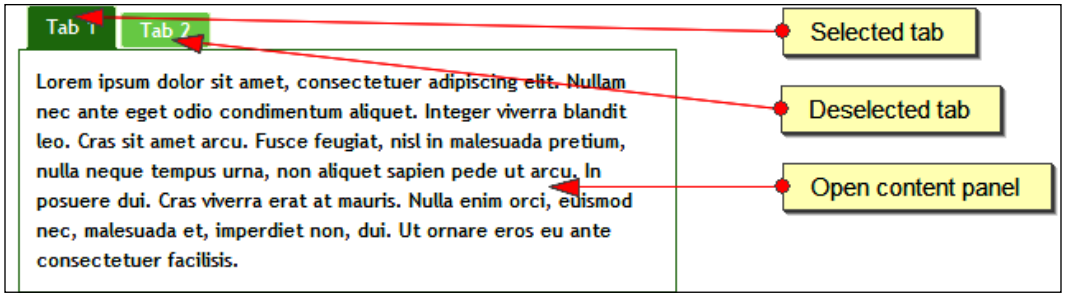
We've covered some important topics during the course of this chapter, but now, thankfully, we can get on with using the components of jQuery UI and get down to some proper coding!

# 2
# Tabs

Now that we've been formally introduced to the jQuery UI library, we can move on to begin looking at the components included in the library. Over the next six chapters, we'll be looking at the widgets provided by the library. These widgets are a set of visually engaging, highly configurable user interface widgets built on top of the foundation provided by the low-level interaction helpers.

The UI tabs widget is used to toggle visibility across a set of different elements; each element containing content can be accessed by clicking on its heading which appears as an individual tab. Each element, or section of content, has a tab that it is associated with and only one of these content sections may be open at a time.

The following image shows the different components of a set of UI tabs:




In this chapter, we will look at the following subjects:

- The default implementation of the widget
- How to style a set of tabs
- Configuring tabs using their properties
- Built-in transition effects for content panel changes
- Controlling tabs using their methods
- Custom events defined by tabs
- AJAX tabs

# A basic tab implementation

The structure of the underlying HTML elements, on which tabs are based, is fairly rigid and widgets require a certain number of elements for them to work.

The tabs themselves must be created from a list element, either ordered or unordered, and each list item should contain a `<span>` element and an `<a>` element. Each link will also need to have a corresponding `<div>` element that it is associated with via its `href` attribute. We'll clarify the exact structure of these elements after our first example.

In a new file in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 1</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //create the tabs
       $("#myTabs").tabs();
     });
    </script>
  </body>
</html>
```

Save the code as `tabs1.html` in your `jqueryui` working folder. Let's review what was used. The following script and CSS resources are needed for the default tab widget instantiation:

- `flora.tabs.css`, `default.all.css`, or a custom stylesheet containing the relevant selectors
- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.tabs.js`

A set of tabbed panels consists of several standard HTML elements arranged in a specific manner (these can be either hardcoded into the page or added dynamically, or a mixture of both depending on the requirements of your implementation):

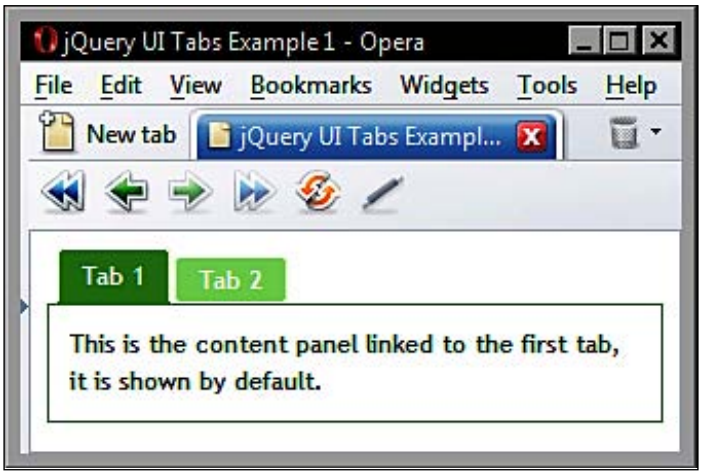
- A list element (`<ul>` or `<ol>`)
- An `<a>` element
- A `<span>` element
- A `<div>` element

The first three elements make the clickable tab headings which are used to open the content section the tab is associated with. Each tab should include a list item containing the link, with a `<span>` element nested inside that link. The `href` attribute of the link should be set as a fragment identifier, prefixed with `#` and should match the `id` attribute of the `<div>` that forms the content section it is associated with.

The content sections of each tab are created by the `<div>` elements. The `id` attribute is required and will be targeted by its corresponding `<a>` element. The elements discussed so far, along with their required attributes, are the minimum that are required from the underlying mark-up.

After the three required script files from the library, we can turn to our custom `<script>` element in which we add the code that creates the tabs. We simply use the `$(function(){})` shortcut to execute our code when the document is ready. We then call the `tabs()` constructor method on the jQuery object representing our tabs container element (the `<ul>` with an `id` of `myTabs`).

The following screenshot shows the default appearance of a tab object:




# Tab styling

The `flora` or `default` stylesheets contain all of the style rules that will make the tabs both appear and function correctly. We could also supply our own stylesheet as long as we provide all of the required selectors. Or, we could produce our own theme using theme roller.

In the following example, we can see how to add basic aesthetic styling. We can override any rules used purely for display purposes with our own simple style rules for quick and easy customization without changing the rules related to the tab functionality.

In a new file in your text editor, create the following very small stylesheet:

```
.ui-tabs-panel {
  width:300px;
  border:1px solid #0000cc;
}
.ui-tabs-nav a, .ui-tabs-nav a span {
  background:url(../img/tab-sprite.gif) no-repeat;
}
.ui-tabs-nav a {
  background-position:100% 0%;
}
```

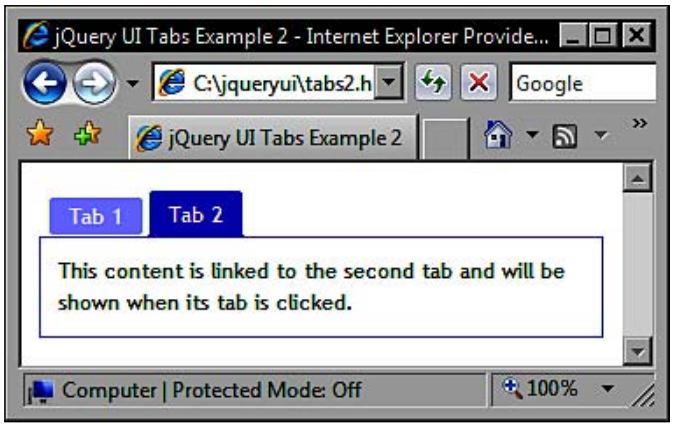
This is all we need. Save the file as `tabsTheme.css` in your `styles` folder. Several class names are automatically added to the elements that make up the tabs and their associated content panels by the widget. We can make use of these class names to change particular features of the widget, such as the background image used to create the tab headings and the borders of the tab content panels.

We've specified a new image with our theme file – `tab-sprite.gif`. This file is used by the widget to give the tab headings the appearance of tab headings. The easiest way to create a new image is to open the existing image used by `flora` in an image editor and then change the colors of the various parts of the image. The original `flora` image can be found in the `i` folder, inside the `themes` folder, of the unpacked library.

Don't forget to link to the new stylesheet from the `<head>` of the underlying HTML file, and make sure the custom stylesheet appears after the `flora.tabs.css` file as shown here:

```
<link rel="stylesheet" type="text/css" href="styles/tabsTheme.css">
```

Once this has been added, save the altered file as `tabs2.html` and review it in a browser. It should look like the following screenshot (with blue instead of green styling):




By overriding just three of the rules defined by the `flora` theme, we can completely change the color scheme of the tabs widget, allowing us to easily integrate tabs into our existing site style.

# Configurable properties

An object can be passed to the `tabs()` constructor method to configure different properties of the tabbed interface. The following table provides the available properties to configure non-default behaviours when using the tabs widget:

| Property | Default Value | Usage |
|---|---|---|
| `ajaxOptions` | `{}` | Options for remote AJAX tabs |
| `cache` | `"false"` | Load remote tab content only once (lazy-load) |
| `cookie` | `null` | Show active tab using cookie data on page load |
| `disabled` | `[]` | Disable specified tabs on pageload |
| `idPrefix` | `"ui-tabs-"` | Used when a remote tab's link element has no `title attribute` |
| `event` | `"click"` | Tabs event that triggers display of content |
| `fx` | `null` | Specify a transition effect when changing tabs |
| `panelTemplate` |  | A string specifying the elements used for the content section of a dynamically created tab |
| `selected` | `0` | The tab selected by default when the widget renders |
| `spinner` | `"Loading&#B230"` | Specify the loading spinner for remote tabs |
| `tabTemplate` | <li><a href="#{href} ">#{label} </a></li> | A string specifying the elements used when creating new tabs dynamically |
| `unselect` | `false` | Hides an already selected tab when it is clicked |

In addition to these properties, default values for all of the class names for the different elements and states of the tab widget are also defined:

| Property | Default Value |
|---|---|
| `disabledCLass` | `"ui-tabs-disabled"` |
| `hideClass` | `"ui-tabs-hide"` |
| `loadingClass` | `"ui-tabs-loading"` |
| `navClass` | `"ui-tabs-nav"` |
| `panelClass` | `"ui-tabs-panel"` |
| `selectedClass` | `"ui-tabs-selected"` |
| `unselectClass` | `"ui-tabs-unselect"` |

We targeted some of these properties when we wrote our custom stylesheet in the previous example. These properties make the widget more flexible in the class names that are automatically applied to different elements within it.

Let's look at how these configurable properties can be used. For example, if we wanted to configure the tabs to initially display the second content panel when the widget is rendered, we could use the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 3</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
    //define config object
    var tabOpts = {
      selected: 1
   };
      //create the tabs
   $("#myTabs").tabs(tabOpts);
     });
   </script>
  </body>
</html>
```

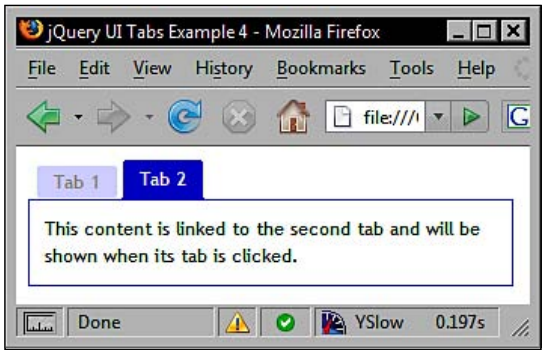
Save this as `tabs3.html` in your `jqueryui` folder. The different tabs, and their associated content panels, are represented by a numerical index starting at zero, much like a standard JavaScript array. Specifying a different tab to open by default is as easy as supplying its index number as the value for the `selected` property. You can also specify that no tabs should open initially by supplying `null` as the value for this property.

You may want a particular tab to be disabled until a certain condition is met. This is easily achieved by manipulating the `disabled` property of the tabs. This property is an empty array by default, but you can disable a tab just by adding its index as an item in this array. Change `tabs3.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 4</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div><script type="text/javascript"
src="jqueryui1.6rc2/jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //define config object
      var tabOpts = {
      selected: 1,
      disabled: [0]
```

```
        };
         //create the tabs
         $("#myTabs").tabs(tabOpts);
       });
    </script>
  </body>
</html>
```

Save this as `tabs4.html` in your `jqueryui` folder. In this example, we added the index of the first tab to the `disabled` array. We could add the indices of other tabs to this array as well, separated by a comma, to disable multiple tabs by default. When the page is loaded in a browser, the first tab will be permanently styled in the hover state and will not respond to mouse overs or clicks at all as seen in this example:




# Transition effects

We can easily add attractive transition effects, which are displayed when tabs open and close, using the `fx` property. This property is configured using another object literal, or an array, inside our configuration object which enables one or more effects. Let's enable fading effects using the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 5</title>
  </head>
```

```
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //define config object
      var tabOpts = {
        fx: {
          opacity: "toggle",
          duration: "slow"
        }
       };
        //create the tabs
        $("#myTabs").tabs(tabOpts);
     });
   </script>
  </body>
</html>
```

Save this file as `tabs5.html` in your `jqueryui` folder. The `fx` object we created has two properties. The first property is the animation. To use fading, we specify `opacity` as this is what is adjusted, to use opening animations we would specify `height` as the property name. Toggling the `opacity` simply reverses its current setting. If it is currently visible, it is made invisible, and vice-versa.

The second property, `duration`, specifies the speed at which the animation occurs. The values for this property are `slow`, `normal`, or `fast`, with `normal` being the default.

As we can see, when we run the file, the tab content slowly fades out as a tab closes and fades in when a new tab opens. Both animations occur during a single tab interaction. To only show the animation once, when a tab closes for example,

we would need to nest the `fx` object within an array. Change the last `<script>` block in `tabs5.html` so that it appears as follows:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){
    //define config object
    var tabOpts = {
   fx: [{
     opacity: "toggle",
     duration: "slow"
   },
   null]
    };
    //create the tabs
    $("#myTabs").tabs(tabOpts);
  });
</script>
```

The closing effect is contained within an object in the first item of the array, and the opening animation is the second. By specifying `null` as the second item, we effectively turn off opening animations.

We can also specify different animations and speeds for opening and closing animations by adding another object as the second array item if we wish instead of `null`. Save this as `tabs6.html` and view the results in your browser.

# Tab events

The tab widget defines a series of useful properties that allow you to add callback functions to easily perform different actions when certain events exposed by the widget are detected. The following table lists the configuration properties that are able to accept executable functions on an event:

| Property | Usage |
|---|---|
| `add` | Execute a function when a new tab is added |
| `disable` | Execute a function when a tab is disabled |
| `enable` | Execute a function when a tab is enabled |
| `load` | Execute a function when a tab's remote data has loaded |
| `remove` | Execute a function when a tab is removed |
| `select` | Execute a function when a tab is selected |
| `show` | Execute a function when the content section of a tab is shown |

Each component of the library has callback properties, such as those in the previous table, which are tuned to look for key moments in any visitor interaction. These properties make it very easy to add code that reacts to different situations. Any functions we use with these callbacks are usually executed *before* the change happens. Therefore, you can return false from your callback and prevent the action from occurring.

The previous technique is the standard means of working with events in the jQuery UI world. There is also a less common way that may become necessary in certain situations.

We can also use the standard jQuery `bind()` method to bind an event handler to a custom event fired by the tabs widget in the same way that we could bind to a standard DOM event, such as a click.

The reason this is possible is that apart from internally invoking any callback function specified in one of the properties listed above, custom events are also fired when different things occur.

The following table lists the tab's custom binding events and their triggers:

| Event | Trigger |
|---|---|
| `tabsselect` | A tab is selected |
| `tabsload` | A remote tab has loaded |
| `tabsshow` | A tab is shown |
| `tabsadd` | A tab has been added to the interface |
| `tabsremove` | A tab has been removed from the interface |
| `tabsdisable` | A tab has been disabled |
| `tabsenable` | A tab has been enabled |

The first three events are fired in succession in the order in which they appear in the table. If no tabs are remote, `tabsselect` and `tabsshow` are fired in that order. These events are all fired *after* the action has occurred. So, the `tabsadd` event will fire after the new tab has been added. In our next example, we can look at how easy it is to react to a particular tab being selected using the standard non-bind technique. Change the `tabs6.html` file so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
```

```
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 7</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
         //define function to be executed on document ready
         $(function(){
            //alert the id of the tab that was selected
            function handleSelect(event, tab) {
               alert("The tab at index " + tab.index + " was
selected");
            }
            //define config object
            var tabOpts = {
               select:handleSelect
            };
            //create the tabs
            $("#myTabs").tabs(tabOpts);
         });
      </script>
  </body>
</html>
```

Save this file as `tabs7.html` in your `jqueryui` folder. We made use of the `select` callback in this example, although the principal is the same for any of the other custom events fired by tabs. The name of our callback function is provided as the value of the `select` property in our configuration object.

Two arguments will automatically be passed to the function we define by the widget when it is executed. These are the original event object, and a custom object containing useful properties from the tab which is in the function's scope.

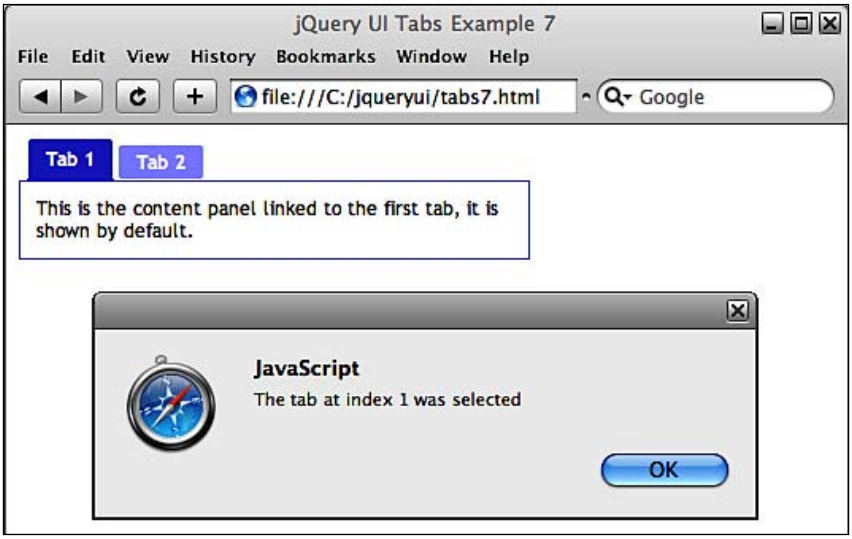
Scope can be a tricky subject, and I'm assuming here that you already have some knowledge of scope in JavaScript. If you don't, the simple explanation for this example is that whichever tab is clicked will be in the scope chain in the context of our callback function.

To tell which of the tabs was clicked, we can look at the `index` property of the second object (remember these are zero-based indices). This is added, along with a little explanatory text, to a standard JavaScript alert.

In this example, the callback function was defined outside of the configuration object, and was instead referenced by the object. We can also define these callback functions inside of our configuration object to make our code more efficient. For example, our function and configuration object from the previous example could have been defined like this:

```
var tabOpts = {
  add: function(event, tab) {
    alert("The tab at index " + tab.index + " was selected");
  }
}
```

See `tabs7inline.html` in the code download for further clarification on this way of using event callbacks. Whenever a tab is selected, you should see the alert before the change occurs as seen below:



# Using tab methods

The tabs widget contains many different methods, which means it has a rich set of behaviours and also supports the implementation of advanced functionality that allows us to work with it programmatically. Let's take a look at these methods which are listed in the following table:

| Method | Usage |
|---|---|
| add | Add a new tab programmatically, specifying the URL of the tab's content, a label, and optionally its index number as arguments |
| remove | Remove a tab programmatically, specifying the index of the tab to remove |
| enable | Enable a disabled tab based on index number |
| disable | Disable a tab based on index number |
| select | Select a tab programmatically, which has the same effect as when a visitor clicks a tab, based on index number |
| load | Reload an AJAX tab's content, specifying the index number of the tab |
| url | Change the URL of content given to an AJAX tab; the method expects the index number of the tab and the new URL |
| destroy | Completely remove the tabs widget |
| length | Return the number of tabs in the widget |
| rotate | Automatically changes the active tab after a specified number of milliseconds have passed either once or repeatedly |

I mentioned jQuery UI's simplified API in Chapter 1, and in the next few examples, we'll get to see just how simple using it is.

# Enabling and disabling tabs

We can make use of the `enable` or `disable` methods to programmatically enable or disable specific tabs. This will effectively switch on any tabs that were initially disabled. Let's use the `enable` method to switch on the first tab, which we disabled by default in an earlier example. Change the `tabs4.html` file as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme.css">
```

```
        <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
        <title>jQuery UI Tabs Example 8</title>
      </head>
      <body>
        <ul id="myTabs">
          <li><a href="#0"><span>Tab 1</span></a></li>
          <li><a href="#1"><span>Tab 2</span></a></li>
        </ul>
        <div id="0">This is the content panel linked to the first tab, it
    is shown by default.</div>
        <div id="1">This content is linked to the second tab and will be
    shown when its tab is clicked.</div>
     <button id="enable">Enable!</button> <button id="disable">Disable!
</button>
        <script type="text/javascript" src="jqueryui1.6rc2/
    jquery-1.2.6.js"></script>
        <script type="text/javascript" src="jqueryui1.6rc2/ui/
    ui.core.js"></script>
        <script type="text/javascript" src="jqueryui1.6rc2/ui/
    ui.tabs.js"></script>
        <script type="text/javascript">
         //define function to be executed on document ready
         $(function(){
             //define config object
          var tabOpts = {
             selected: 1,
             disabled: [0]
          };
           //create the tabs
           $("#myTabs").tabs(tabOpts);
       //set click hander for the enable button
       $("#enable").click(function() {
         //enable the first tab
         $("#myTabs").tabs("enable", 0);
       });
       //set click handler for the disable button
       $("#disable").click(function() {
         //disbale the second tab
         $("#myTabs").tabs("disable", 1);
       });
          });
       </script>
      </body>
    </html>
```

Save the changed file as `tabs8.html` in your `jqueryui` folder. We use the `click` event of the enable button to call the `tabs` constructor once more. This passes in the string `"enable"` which specifies the `enable` method and the index number of the tab we want to enable. The `disable` method is used in exactly the same way. You'll see that the second tab cannot be disabled until the first tab has been enabled and selected.

All methods exposed by each component are used in this same easy way which you'll see more of as we progress through the book.

# Adding and removing tabs

As well as enabling and disabling tabs programmatically, we can also remove or add completely new tabs dynamically just as easily. Change `tabs8.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
 <link rel="stylesheet" type="text/css" href="styles/tabsTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 9</title>
  </head>
  <body>
 <div>
      <ul id="myTabs">
        <li><a href="#0"><span>Tab 1</span></a></li>
        <li><a href="#1"><span>Tab 2</span></a></li>
      </ul>
      <div id="0">This is the content panel linked to the first tab,
it is shown by default.</div>
      <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
 </div>
 <label>Enter a tab to remove:</label><input id="indexNum"><button
id="remove">Remove!</button><br><br>
 <button id="add">Add a new tab!</button><br><br>
 <div id="newTab">This content will become part of the tabs when the
above button is clicked!</div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //create the tabs
       $("#myTabs").tabs();
   //add click handler for 'remove' button
   $("#remove").click(function() {
     //get the index to remove
     var indexNumber = $("#indexNum").val() - 1;

     //remove the tab
     $("#myTabs").tabs("remove", indexNumber);
   });
   //add click handler for 'add' button
   $("#add").click(function() {
     //define tab label
     var newLabel = "A New Tab!"
     //add the new tab
     $("#myTabs").tabs("add", "#newTab", newLabel);
    });
      });
   </script>
  </body>
</html>
```

Save this as `tabs9.html` in your `jqueryui` folder. The first change is that we're now linking to a new stylesheet called `tabsTheme2.css`. This is identical to the first custom theme we used, but with the following selectors and rules:

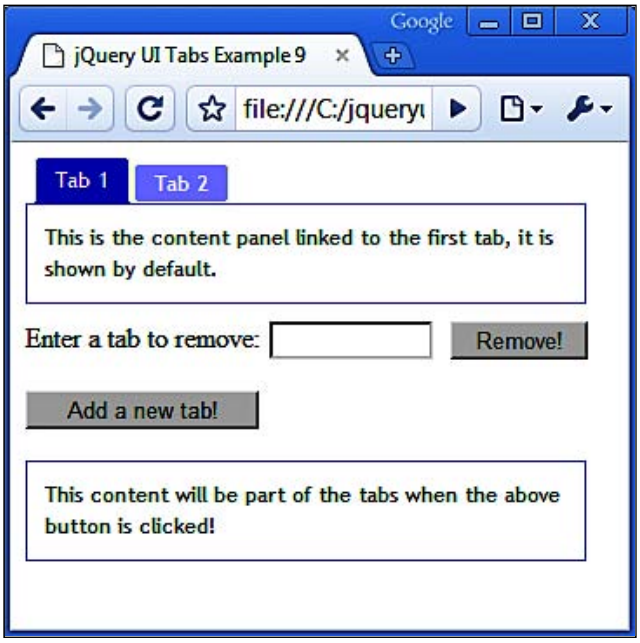
```
label { float:left; width:140px; }
input, button {
  float:left; display:block; width:90px; margin-right:10px;
}
button { width:79px; }
#add { width:134px; }
```

Apart from some new JavaScript which is required to invoke the `add` method, we also need to encase our tabs and their associated content within a container `<div>`. Without this, the new tab will be added as the last element in the `<body>` instead of the last element in the tabs widget.

The first of our new functions handles removing a tab using the `remove` method. This method requires one additional argument which is the index number of the tab to be removed. In this example, we get the value entered into the text box and pass it as the argument.

The `add` method, which adds a new tab to the widget, can be made to work in several different ways. In this example, we've specified that existing content already on the page (the `<div>` with an `id` of `newTab`) should be added to the tabs widget. In addition to passing the string `"add"`, and specifying a reference to the element we wish to add to the tabs, we also specify a label for the new tab.

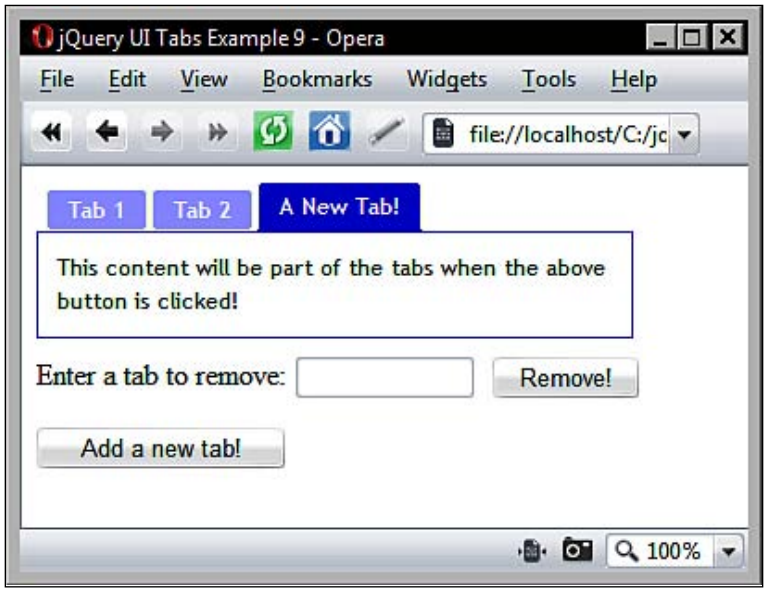
Optionally, we can also specify the index number where the new tab should be inserted. If the index is not supplied, the new tab will be added as the last tab. When you run the page in a browser, you should see that although the `<div>` we have specified is added to the tabs interface, it doesn't automatically pick up the styling of the rest of the widget. It is initially visible before it is added to the widget, as shown in the following screenshot:




There are several easy ways in which this can be fixed. If the tab content does not need to be visible initially, we can simply add the appropriate class names to the content's container element:

```
<div id="newTab" class="ui-tabs-panel ui-tabs-hide">This content will be
part of the tabs when the above button is clicked!</div>
```

Now when the page loads, this content will not be visible, and will remain hidden until it has been added to the tabs and its tab has been selected as seen below:




If the content does need to be shown when the page initially loads, or if it is not known which elements on the page will be added to the tabs, it is simple enough to add these classes to the tab content `<div>` when the button is clicked.

# Simulating clicks

There may be times when you want to programmatically select a particular tab and show its content. This could happen as the result of some other interaction by the visitor. To do this, we can use the `select` method, which is completely analogous with the action of clicking a tab. Alter the final `<script>` block in `tabs9.html` so that it appears as follows:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){
    //create the tabs
    $("#myTabs").tabs();
    //add click handler for 'remove' button
    $("#remove").click(function() {
      //get the index to remove
      var indexNumber = $("#indexNum").val() - 1;
      //remove the tab
```

```
        $("#myTabs").tabs("remove", indexNumber);
    });
     //add click handler for 'add' button
     $("#add").click(function() {
       //define tab label
      var newLabel = "A New Tab!"
       //add the new tab
       $("#myTabs").tabs("add", "#newTab", newLabel);

    //new tab will be at end, get index
    var newIndex = $("#myTabs").tabs("length") - 1;

    //select the new tab
    $("#myTabs").tabs("select", newIndex);

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

Save this as `tabs10.html` in your `jqueryui` folder. Now when a new tab is added, it is automatically selected. The `select` method requires just one additional parameter which is the index number of the tab to select.

As any tab we add will be the last tab in the interface, and as the tab indices are zero based, all we have to do is use the `length` method to return the number of tabs and then subtract 1 from this figure. The result is passed to the `select` method.

# Creating a tab carousel

The last method that we'll look at in this chapter is the `rotate` method. The `rotate` method will make all of the tabs, and their associated content panels, display one after the other automatically. It's a great visual effect and is useful for ensuring that all of the tab content gets seen by the visitor. For an example of this kind of effect in action, see the homepage of `http://www.cnet.com`.

Like the other methods we've seen, the `rotate` method is extremely easy to use. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme2.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 11</title>
  </head>
  <body>
    <div>
      <ul id="myTabs">
        <li><a href="#0"><span>Tab 1</span></a></li>
        <li><a href="#1"><span>Tab 2</span></a></li>
      </ul>
      <div id="0">This is the content panel linked to the first tab,
it is shown by default.</div>
      <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
         //define function to be executed on document ready
         $(function(){

            //create the tabs and make them rotate
            $("#myTabs").tabs().tabs("rotate", 1000, true);
         });
      </script>
  </body>
</html>
```

Save this file as `tabs11.html` in your `jqueryui` folder. Although we can't call the `rotate` method directly using the initial `tabs` constructor method, we can chain it to the end like we would methods from the standard jQuery library.

The `rotate` method is used with two additional parameters. The first parameter is an integer which specifies the number of milliseconds each tab should be displayed before the next tab is shown. The second parameter is a boolean which indicates whether the cycle through the tabs should occur once or continuously.



**Chaining Widgets**

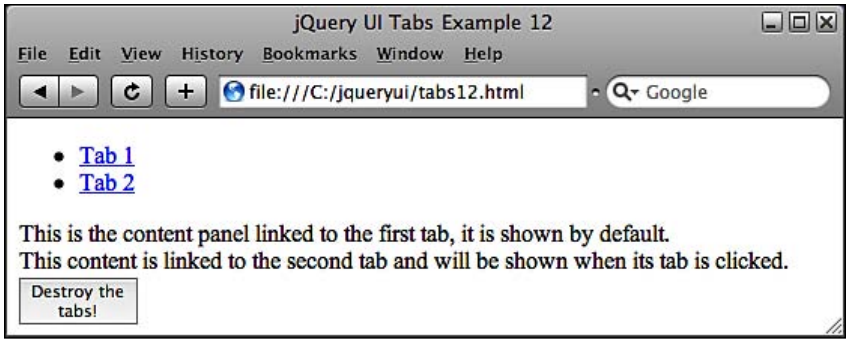
Chaining widget methods is possible because like the methods found in the underlying jQuery library, they always return the jQuery object.

The tab widget also contains a `destroy` method. This is a method that is common to all of the widgets found in jQuery UI. Let's see how it works. Create another new page and add to it the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 12</title>
  </head>
  <body>
    <div>
      <ul id="myTabs">
        <li><a href="#0"><span>Tab 1</span></a></li>
        <li><a href="#1"><span>Tab 2</span></a></li>
      </ul>
      <div id="0">This is the content panel linked to the first tab,
it is shown by default.</div>
      <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    </div>
      <button id="destroy">Destroy the tabs!</button>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
         //define function to be executed on document ready
         $(function(){

            //create the tabs
            $("#myTabs").tabs();

            //add click handler for button
            $("#destroy").click(function() {

               //destroy the tabs
               $("#myTabs").tabs("destroy");
            });
```

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

Save this file as `tabs12.html` in your `jqueryui` folder. The `destroy` method, which we invoke with a click on the `<button>`, completely removes the tab widget, returning the underlying HTML to its original state. After the button has been clicked, you should see a standard HTML list element and the text from each tab, just like in the following screenshot:




# AJAX tabs

We've looked at adding new tabs from existing content already on the page, in addition to this, we can also create AJAX tabs that load content from remote files or URLs. Let's extend our example of adding tabs from earlier so that the new tab content is loaded from an external file. In a new page in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 13</title>
  </head>
  <body>
    <div>
      <ul id="myTabs">
```

```
        <li><a href="#0"><span>Tab 1</span></a></li>
        <li><a href="#1"><span>Tab 2</span></a></li>
      </ul>
      <div id="0">This is the content panel linked to the first tab,
it is shown by default.</div>
      <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    </div>
    <button id="add">Add a new tab!</button>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create the tabs
       $("#myTabs").tabs();

       //add click handler for 'add' button
       $("#add").click(function() {

         //define tab label
         var newLabel = "A New Tab!"

         //add the new tab
         $("#myTabs").tabs("add", "tabContent.html", newLabel);
       });
     });
   </script>
  </body>
</html>
```

Save this as `tabs13.html` in your `jqueryui` folder. This time, instead of specifying an element selector as the second argument of the `add` method, we supply a relative file path. Instead of generating an in-page tab from the specified element, the tab becomes an AJAX tab and loads the contents of the remote file.

You probably won't notice it when running this file locally, but when your page is up online, you'll see that while the tab is loading its remote content the configured spinner will be displayed. By default, this appears as **loading...**.

The file used as the remote content in this example is extremely basic and consists of just the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>Content Page</title>
  </head>
  <body>
    <div>This is some new content!</div>
  </body>
</html>
```

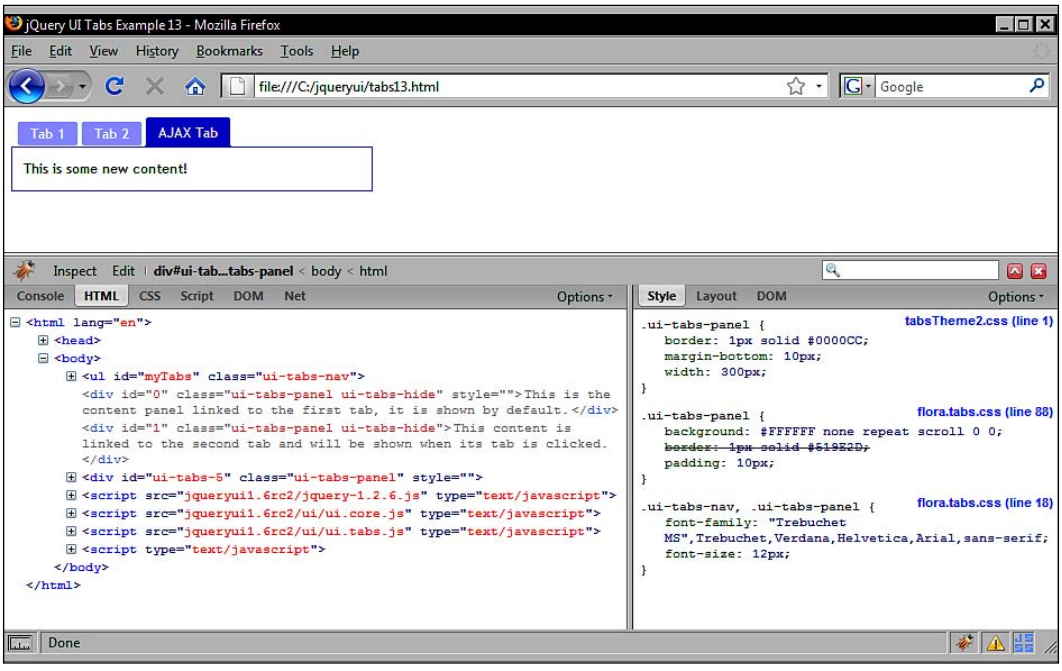
Save this as `tabContent.html` in your `jqueryui` folder. Instead of using JavaScript to add the new tab in this way, we can use plain HTML to specify an AJAX tab as well. In this example, we want the tab which will display the remote content to be available all the time, not just after clicking a button. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 14</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
      <li><a href="tabContent.html"><span>AJAX Tab</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create the tabs
       $("#myTabs").tabs();
     });
   </script>
  </body>
</html>
```

Save this as `tabs14.html` in your `jqueryui` folder. All we do is specify the path to the remote file (the same one we created in the previous example) using the `href` attribute of a link element in the mark-up from which the tabs are created.

Unlike static tabs, we don't need a corresponding `<div>` element with an `id` that matches the `href` of the link. The additional elements required for the tab content will be generated automatically by the widget. We also don't need to wrap our tabs in an outer container `<div>`. Here's what the widget will look like:



I've included the Firebug console in the previous screenshot so you can see where in the DOM of the page the new tab content is added.

You should note that there is no inherent cross-domain support built into the AJAX functionality of the tabs widget. So, unless additional PHP, or some other server-scripting language, is employed as a proxy, or you wish to make use of JSON structured data, files and URLs should be under the same domain as the page running the widget.

As well as loading data from external files, it can also be loaded from URLs. This is great when retrieving content from a database using query strings. Methods related to AJAX tabs include the `load` and `url` methods. The `load` method is used to load and reload the contents of an AJAX tab, which could come in handy for refreshing content that changes very frequently.

The `url` method is used to change the URL where the AJAX tab retrieves its content. Let's look at a brief example of these two methods in action. Change `tabs13.html` so that it matches the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
tabsTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Tabs Example 15</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
      <li><a href="tabContent.html"><span>AJAX Tab</span></a></li>
    </ul>
    <div id="0">This is the content panel linked to the first tab, it
is shown by default.</div>
    <div id="1">This content is linked to the second tab and will be
shown when its tab is clicked.</div>
 <select id="fileChooser">
   <option>tabContent.html</option>
   <option>tabContent2.html</option>
 </select>
```

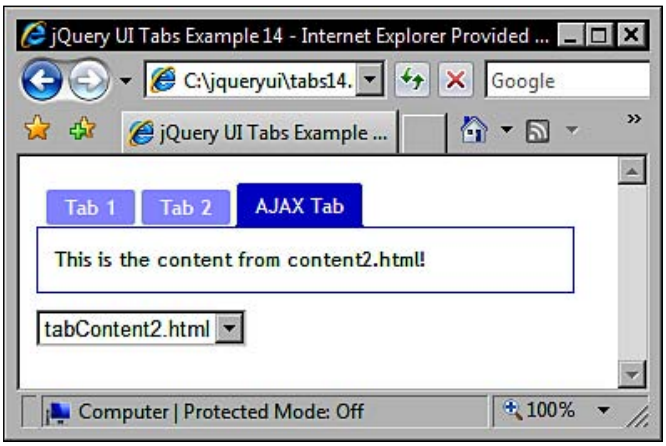
```
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create the tabs
       $("#myTabs").tabs();

   //define handler for change event on select element
   $("#fileChooser").change(function() {

     //load either file 1 or file 2
     this.selectedIndex == 0 ? loadFile1() : loadFile2();

     //load the new file
     function loadFile1() {
      $("#myTabs").tabs("url", 2, "tabContent.html").tabs("load", 2);
     }
     function loadFile2() {
      $("#myTabs").tabs("url", 2, "tabContent2.html").tabs("load", 2);

     }

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

Save the new file as `tabs15.html` in your `jqueryui` folder. We've added a simple `<select>` element to the page that lets you choose the content to display in the AJAX tab. In the JavaScript, we set a `change` handler for the `<select>` and specified an anonymous function to be executed each time the event is detected.

This function checks the `selectedIndex` of the `<select>` element and calls either the `loadFile1` or `loadFile2` function. The `<select>` element is in the execution scope of the function so we can refer to it using the `this` keyword.

These functions are where things gets interesting. We first call the `url` method, specifying two additional arguments which are the index of the tab whose URL we want to change followed by the new URL. We then call the `load` method, which is chained to the `url` method, specifying the index of the tab whose content we want to load.

You can run the new file in your browser and select the AJAX tab. Then use the `<select>` to choose the second file and watch as the content of the tab is changed as seen here:




You'll also see that the tab content will be reloaded even if the AJAX tab isn't selected when you use the select element.

# Fun with tabs

Let's pull in some external content for our final tabs example. If we use the tabs widget, in conjunction with the standard jQuery library `getJSON` method, we can by-pass the cross-domain exclusion policy and pull in a feed from another domain to display in a tab. In a new file in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
flickrTabTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI AJAX Tabs Example</title>
  </head>
  <body>
    <div>
      <ul id="myTabs">
        <li><a href="#0"><span>Nebula Information</span></a></li>
```

```
        <li><a href="#flickr"><span>Images</span></a></li>
      </ul>
      <div id="0">
        <p>A nebulae...</p>
      </div>
      <div id="flickr"></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
  </body>
</html>
```

The HTML seen here is nothing new. It's basically the same as the previous examples so I won't describe it in any detail. The only point worthy noting is that unlike the previous AJAX tab examples, we have specified an empty `<div>` element which will be used for the AJAX tab's content. Now, just before the `</body>` tag, add the following script block:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){

    //create config object
    var tabOpts = {
      select: function(event, ui) {

      //see if flickr tab selected
      ui.tab.toString().indexOf("flickr") != -1 ? getData() : null ;

        //define getData function
        function getData() {

          //get rid of any previous images
          $("#flickr").empty();

          //get JSON feed from flickr
                      $.getJSON("http://api.flickr.com/services/
feeds/photos_public.gne?tags=nebula&format=json&jsoncallback=?",
function(data) {

            //iterate over each object in JSON feed
            $.each(data.items, function(i,item){

              //create and format new image
```

```
                $("<img/>").attr("src", item.media.
m).appendTo("#flickr").height(100).width(100).css({
                  marginRight:"5px",
                  marginBottom:"5px",
                  borderColor:"#000000",
                  borderStyle:"solid",
                  borderWidth:"1px"
                });

                //stop after 6 images
                if (i == 5) return false;
              });
            });
          }
        }
      }

      //create the tabs
      $("#myTabs").tabs(tabOpts);
    });
</script>
```

Every time a tab is selected, our `select` callback will check to see if it was the tab with an `id` of `flickr` that was clicked. If it is, then the `getData()` function is invoked which uses the standard jQuery `getJSON` method to retrieve an image feed from `http://www.flickr.com`.

Once the data is returned, the anonymous callback function iterates over each object within the feed and creates a new image. We also remove any pre-existing images from the content panel to prevent a build-up of images following multiple tab selections.

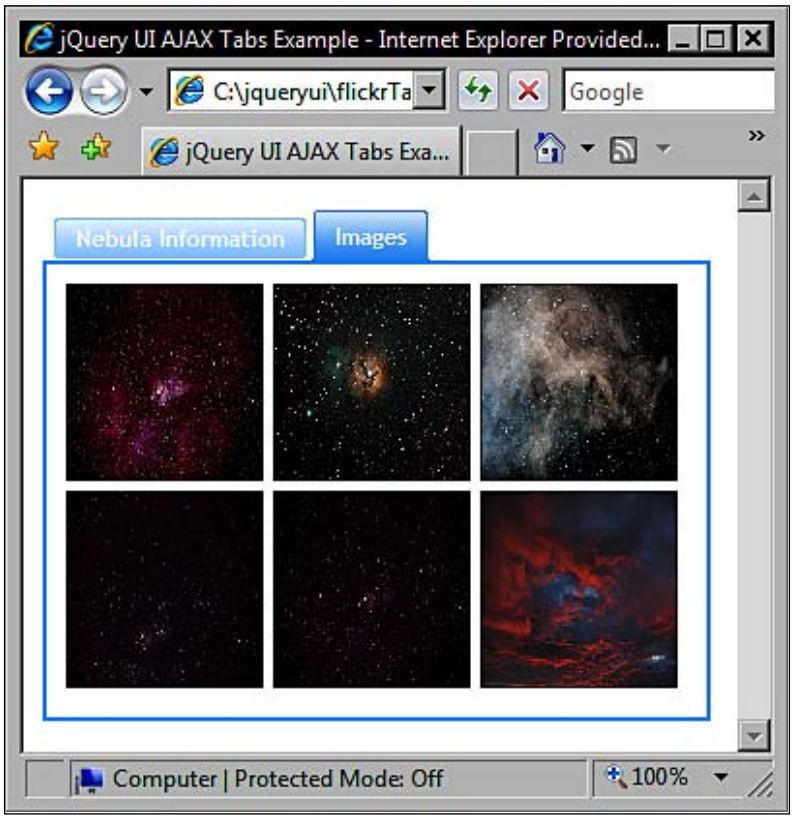
Each new image has its `src` attribute set using the information from the current feed object. It then has its dimensions and various other CSS properties set. Lastly, it is added to the empty Flickr tab. Once iteration over six of the objects in the feed has occured, we exit the iteration. It's that simple. Save the file as `flickrTab.html` in your `jqueryui` folder.

We're also using a different stylesheet, in addition to the `flora` theme for the tabs widget, in this example. Again, this is similar to other stylesheets we've used in this section:

```
.ui-tabs-panel {
  width:321px;
  border:2px solid #3399ff;
}
.ui-tabs-nav a, .ui-tabs-nav a span {
```

```
    background:url(../img/flickr-tab/headerSprite.gif) no-repeat;
}
.ui-tabs-nav a {
    background-position:100% 0%;
}
```

Save this as `flickrTabTheme.css` in your `styles` folder. When you view the page and select the `Images` tab, after a short delay you should see six new images, as seen in the following screenshot:



# Summary

The tabs widget is an excellent way of saving space on your page by organizing related, or even completely unrelated) sections of content that can be shown, or hidden, with simple click-input from your visitors. It also lends an air of interactivity to your site that can help improve the overall functionality and appeal of the page on which it is used.

Let's review what was covered in this chapter. We first looked at how, with just a little underlying HTML and a single line of jQuery-flavoured JavaScript code, we can implement that default tabs widget.

We then saw how easy it is to add our very own basic styling for the tabs widget so that its appearance, but not its behaviour, is completely altered. We already know that in addition to this there are two stylesheets from jQuery UI we can use (`flora` and `default`), or that we can create a completely new theme using Theme Roller.

We then moved on to look at the set of configurable properties exposed by the tabs API. With these, we can enable or disable different options that the widget supports, such as whether tabs are selected by clicks or another event, whether certain tabs are disabled when the widget is rendered, etc.

We took some time to look at how we can use a range of predefined callback properties that allow us to execute arbitrary code when different events are detected. We also saw that the jQuery `bind()` method can listen for the same events if it becomes necessary.

Following the configurable properties, we looked at the range of methods that we can use to programmatically make the tabs perform different actions, such as simulating a click on a tab, enabling or disabling a tab, and adding or removing tabs.

We briefly looked at some of the more advanced functionality supported by the tabs widget such as AJAX tabs and the tab carousel. Both of these techniques are incredibly easy to use and can add value to any implementation.

# 3
# The Accordion Widget

The accordion widget is another UI widget made up of a series of containers for your content, all of which are closed except for one. Therefore, most of its content is initially hidden from view, much like the tabs widget that we looked at in the previous chapter.

Each container has a heading element associated with it, which is used to open the container and display the content. When you click on a heading, its content is displayed. When you click on another heading, the currently visible content is hidden while the new content is shown.

The accordion widget is a robust and highly configurable widget that allows you to save space on your web pages by only displaying a certain section of related content at any one time. This is like a tabbed interface but positioned vertically instead of horizontally.

It's easy for your visitors to use and it's easy for us to implement. It has a range of configurable properties that can be used to customize its appearance and behaviour. It also has a series of methods that allow you to control it programmatically.

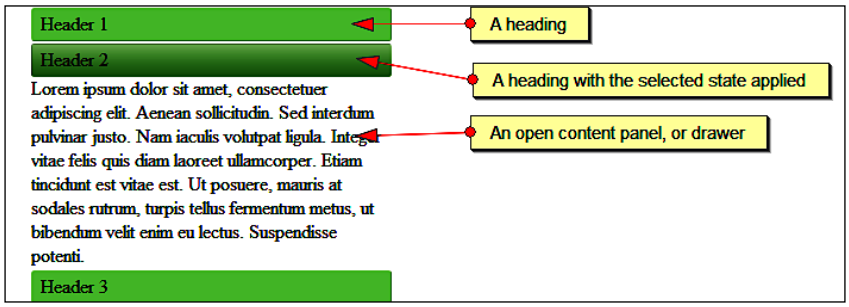
You should note that the height of the accordion's container element will automatically be set so that there is room to show the tallest content panel in addition to the headers. This will vary, of course, depending on the width that you set on the widget's container.

In this chapter, we are going to cover the following topics:

- The structure of an accordion widget
- A default implementation of an accordion
- Adding custom styling
- The configurable properties
- Built-in methods for working with the accordion
- Built-in types of animation
- Custom accordion events

# Accordion's structure

Let's take a moment to familiarize ourselves with what an accordion is made of. Within the outer container is a series of links. These links are the headings within the accordion and each heading will have a corresponding content panel, or drawer as they are sometimes referred to, which opens when the heading is clicked. The following screenshot shows these elements as they may appear in an accordion:

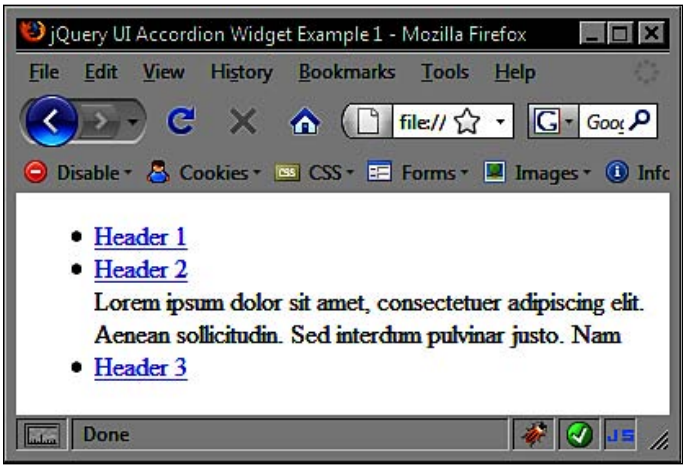



It's worth remembering that when using the accordion widget, only one content panel can be open at any one time. Let's implement a basic accordion now. In a blank page in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 1</title>
  </head>
  <body>
    <ul id="myAccordion">
      <li>
        <a href="#">Header 1</a>
        <div>Wow, look at all this content that can be shown or hidden
with a simple click!</div>
      </li>
      <li>
        <a href="#">Header 2</a>
        <div>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aenean sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. </div>
```

```
      </li>
      <li>
        <a href="#">Header 3</a>
        <div>Donec at dolor ac metus pharetra aliquam. Suspendisse
purus. Fusce tempor ultrices libero. Sed quis nunc. Pellentesque
tincidunt viverra felis. Integer elit mauris, egestas ultricies,
gravida vitae, feugiat a, tellus.</div>
      </li>
    </ul>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //turn specified element into an accordion
       $("#myAccordion").accordion();
     });
   </script>
  </body>
</html>
```

Save the file as `accordion1.html` in your `jqueryui` folder and try it out in a browser. We haven't specified any styling at all at this stage, but as you can see from the following screenshot, it still functions exactly as intended:



Little code is required for a basic working version of the accordion widget. A simple unordered list element is the mark-up foundation which is transformed by the library into the accordion object.

The following three separate external script files are required for an accordion:

- The jQuery library itself (`jquery-1.2.6.js`)
- The UI base file (`ui.core.js`)
- The accordion source file (`ui.accordion.js`)

The first two files are mandatory requirements of all components of the UI library. They should be linked to in the order shown here. Each widget also has its own source file, and may depend on other components as well.

The order in which these files appear is important. The jQuery library must always appear first, followed by the UI base file. After these files, any other files that the widget depends upon should appear before the widget's own script file. The library components will not function as expected if files are not loaded in the correct order.

Finally, we use a custom `<script>` block to turn our `<ul>` element into the accordion. We can use the jQuery object shortcut `$` to specify an anonymous function which will be executed as soon as the document is ready. This is analogous to using `$(document).ready(function(){})` and helps to cut down on the amount of code we have to type.

Following this, we use the simple `ID` selector `$("#myAccordion")` to specify the element on the page we want to transform. We then use the `accordion()` constructor method to create the accordion.

Other elements can be turned into accordions as well. All list element variants are supported including ordered lists and definition lists. You don't even need to base the accordion on a list element at all. You can build a perfectly functional accordion using just nested `<div>` and `<a>` elements, although additional configuration will be required.

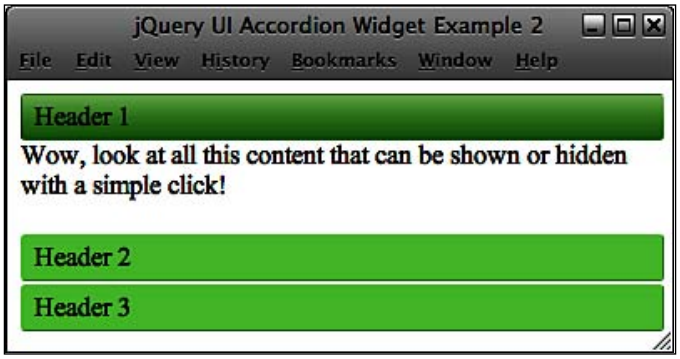
In this example, we used an empty fragment (`#`) as the value of the `href` attribute. You should note that any URLs supplied for accordion headers will not be followed when the header is clicked within the accordion when using the default implementation.

# Styling the accordion

With no styling, the accordion will take up 100% of the width of its container. Like with other widgets, we have several options for styling the accordion. We can create our own custom stylesheet to control the appearance of the accordion and its content, we can use the `default` or `flora` themes that come with the library, or we can use Theme Roller to create an extensive skin for the whole library. Let's see how using the `flora` theme for the accordion will cause it to render. In `accordion1.html`, add the following `<link>` tag to the `<head>` of the page:

```
<link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/themes/
flora/flora.accordion.css">
```

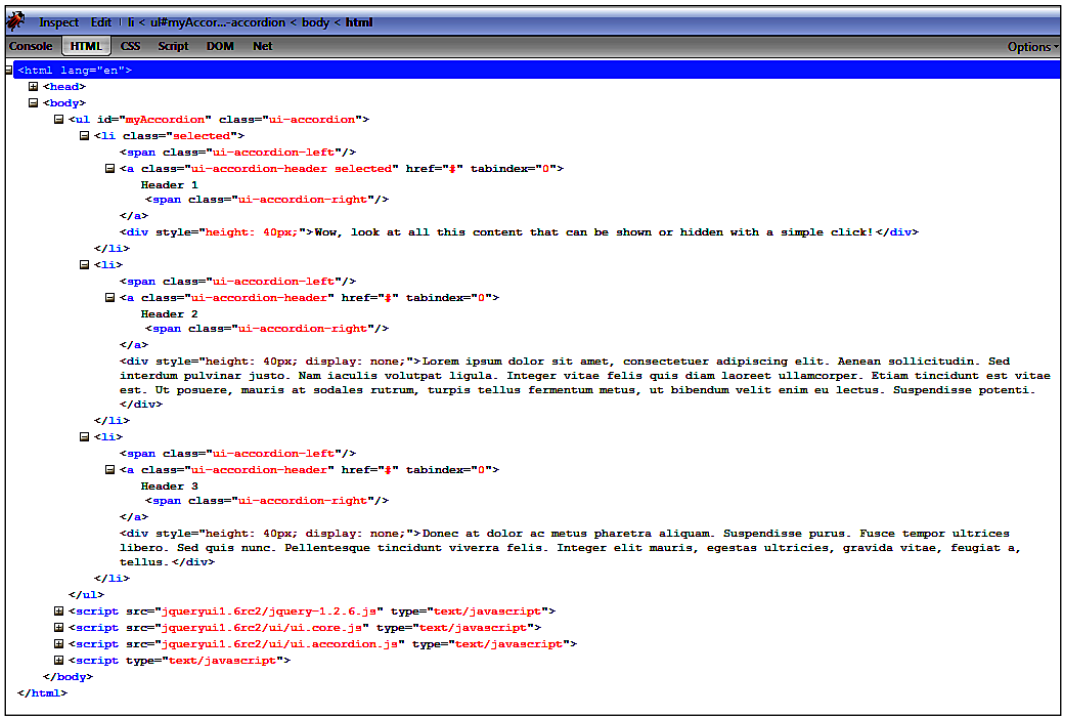
Save the new file as `accordion2.html`, also in the `jqueryui` folder, and view it again in a browser. It should appear something like this:




The accordion theme file assumes that an unordered list is being used as the basis of the widget and specifically targets `<li>` elements with certain style rules. We can easily create our own custom theme to style the accordion for situations where we want to use a non list-based accordion widget, or if we simply want different colors or font styles.

You can use the excellent Firebug plugin for Firefox, or another DOM viewer, to see the class names that are automatically added to certain elements when the accordion is generated. You can also read through an un-minified version of the source file if you really feel like it. These will be the class names that we'll be targeting with our custom CSS.

The following screenshot shows Firebug in action:




Change `accordion2.html` so that it appears as follows (new code is shown in bold):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 3</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum...</div></div>
```

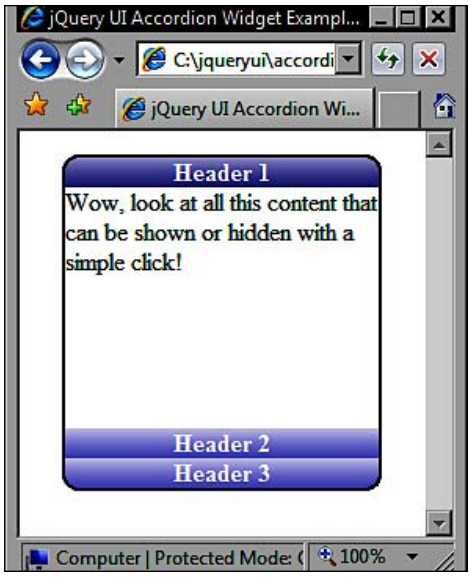
```
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus.</div></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //turn specified element into an accordion
       $("#myAccordion").accordion();
     });
    </script>
  </body>
</html>
```

Save this version as `accordion3.html` in the `jqueryui` folder. The class name `ui-accordion` is automatically added to the accordion's container element. Therefore, we can use this as a starting point for most of our CSS selectors. The links that form our drawer headers are given the class `ui-accordion-header` so we can also target this class name. In a new file, create the following stylesheet:

```
#myAccordion {
  width:200px;
  border:2px solid #000000;
  position:relative;
  list-style-type:none;
  padding-left:0;
}
.ui-accordion-header {
  text-decoration:none;
  font-weight:bold;
  color:#000000;
  display:block;
  width:100%;
  text-align:center;
}
.ui-accordion div div {
  font-size:90%;
}
```

```
.ui-accordion a {
  color:#ffffff;
  background:url(../img/accordion/header-sprite.gif) repeat-x 0px 0px;
}
.ui-accordion a.selected {
  background:url(../img/accordion/header-sprite.gif)
repeat-x 0px -22px;
}
.ui-accordion a:hover {
  background:url(../img/accordion/header-sprite.gif)
repeat-x 0px -44px;
}

/* container rounded corners */
.corner {
  position:absolute;
  width:12px; height:13px;
  background:url(../img/accordion/corner-sprite.gif) no-repeat;
}
.topLeft {
  top:-2px; left:-2px;
  background-position:0px 0px;
}
.topRight {
  top:-2px; right:-2px;
  background-position:0px -13px;
}
.bottomRight {
  bottom:-2px; right:-2px;
  background-position:0px -26px;
}
.bottomLeft {
  bottom:-2px; left:-2px;
  background-position:0px -39px;
}
```

Save this file as `accordionTheme.css` in your `styles` folder and preview `accordion3.html` in a browser. We will need a new folder for the images we use in this and subsequent examples. Create a new folder inside the `img` folder and name it `accordion`. With just two images, and a few simple style rules, we can drastically change the default appearance of the accordion with our own custom skin as shown in the following screenshot:




# Configuring accordion

The accordion has a range of configurable properties which allow us to easily change the default behaviour of the widget. The following table lists the available properties, their default value, and gives a brief description of their usage:

| Property | Default Value | Usage |
|---|---|---|
| `active` | first child | Selector for the initially open drawer |
| `alwaysOpen` | `true` | Ensure that one drawer is open at all times |
| `animated` | `"slide"` | Animate the opening of drawers |
| `autoHeight` | `true` | Automatically set height according to the biggest drawer |
| `clearStyle` | `false` | Clear styles after an animation |
| `event` | `"click"` | Event on headers that trigger drawers to open |
| `fillSpace` | `false` | Accordion completely fills height of its container |
| `header` | `"a"` | Selector for header elements |
| `navigation` | `false` | Enable navigation for accordion |
| `navigationFilter` | location.href | By default, this property opens the drawer whose heading's `href` matches `location.href` |
| `selectedClass` | `"selected"` | Class name applied to headers with open drawers |



**Configurable properties**

The configurable properties for all of the different components of jQuery UI are constantly evolving with each new release of the library. You can keep track of the latest properties by looking through the online jQuery UI API pages. Each component has its own page and can be accessed from `http://docs.jquery.com/UI/`.

Most of the properties are self-explanatory, and the values they accept are usually booleans, strings, or element references. Let's put some of them to use so we can explore their functionality. Alter `accordion3.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 4</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple mouseover!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus.</div></div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //set the event property
      var accOpts = {
        event:"mouseover"
      }

      //turn specified element into an accordion
       $("#myAccordion").accordion(accOpts);

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

First, we create a new object literal called `accOpts` which contains one property key and a value. We then pass this object into the `accordion()` constructor as an argument, and it overrides the default properties of the widget. The string we specified for the value of the `event` property becomes the event that triggers the activation of the drawers, making this a very useful property. Save the changes as `accordion4.html`.

You should note that you can also set properties using an inline object within the widget's constructor method without creating a separate object (see `accordion4Inline.html`). Using the following code would be equally as effective, and would often be the preferred way for coding:

```
<script type="text/javascript">
  //function to execute when doc ready
  $(function() {

    //turn specified element into an accordion
    $("#myAccordion").accordion({
      event:"mouseover"
    });
  });
</script>
```

We can set other properties at the same time as well. If we want to change which drawer is open by default when the accordion is rendered, as well as change the trigger event, we would supply both properties and the required values, with each pair separated by a comma. Update `accordion4.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 5</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a id="header1" href="#">Header 1</a><div>Wow, look at all
this content that can be shown or hidden with a simple mouseover!
</div></div>
      <div><a id="header2" href="#">Header 2</a><div>Lorem ipsum dolor
sit amet, consectetuer adipiscing elit. Aenean sollicitudin. Sed
interdum pulvinar justo. Nam iaculis volutpat ligula. Integer vitae
felis quis diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut
posuere, mauris at sodales rutrum, turpis tellus fermentum metus, ut
bibendum velit enim eu lectus. Suspendisse potenti.</div>
</div>
      <div><a id="header3" href="#">Header 3</a><div>Donec at dolor
ac metus pharetra aliquam. Suspendisse purus. Fusce tempor ultrices
libero. Sed quis nunc. Pellentesque tincidunt viverra felis. Integer
elit mauris, egestas ultricies, gravida vitae, feugiat a, tellus.
</div></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //configure accordion
```

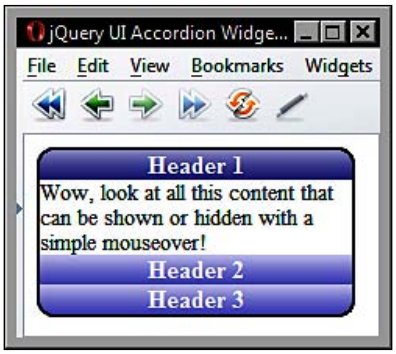
```
        var accOpts = {
          event:"mouseover",
          active:"#header3"
        }

        //turn specified element into an accordion
         $("#myAccordion").accordion(accOpts);

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

The first change is to give our header elements `id` attributes in the underlying HTML in order to target them with the `active` property. In our object literal, we then specify the selector for the header we would like to open by default. Save the file as `accordion5.html`. When the page is opened, the third drawer should be displayed by default.

The other properties listed in the table at the start of this section are equally as easy to configure. Change the object literal so that it appears as follows:

```
//configure accordion
var accOpts = {
  event:"mouseover",
  active:"#header3",
  alwaysOpen:false,
  autoHeight:false
}
```

Save these changes as `accordion6.html` and view the results in a browser. First, you should find that when you first roll over a heading the drawer opens as normal, but the accordion grows or shrinks depending on how much content is in the drawer. It no longer stays at a fixed height. This can be seen in the following example:



You should also find that if you roll over a heading whose drawer is already open, the drawer will close and the accordion will shrink so that only the headers are displayed with no open drawers. Note that when using `false` with the `alwaysOpen` property, the accordion will shrink in this way regardless of whether the `autoHeight` property is set to `true` or `false`.




The `fillSpace` property, if set, will override `autoHeight`. You should also be aware that the `clearStyle` property will not work with `autoHeight`. One final property we should look at is the `navigation` property. This property is used to enable navigating to new pages from accordion headings. Change `accordion6.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 7</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a id="header1" href="#1">Header 1</a><div>Wow, look at all
this content that can be shown or hidden with a simple mouseover!
</div></div>
      <div><a id="header2" href="#2">Header 2</a><div>Lorem ipsum
dolor sit amet, consectetuer adipiscing elit. Aenean sollicitudin. Sed
interdum pulvinar justo. Nam iaculis volutpat ligula. Integer vitae
felis quis diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut
```

```
posuere, mauris at sodales rutrum, turpis tellus fermentum metus, ut
bibendum velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a id="header3" href="#3">Header 3</a><div>Donec at dolor
ac metus pharetra aliquam. Suspendisse purus. Fusce tempor ultrices
libero. Sed quis nunc. Pellentesque tincidunt viverra felis. Integer
elit mauris, egestas ultricies, gravida vitae, feugiat a, tellus.
</div></div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //configure accordion
      var accOpts = {
        event:"mouseover",
        active:"#header3",
        alwaysOpen:false,
        autoHeight:false,
        navigation:true
      }

      //turn specified element into an accordion
       $("#myAccordion").accordion(accOpts);

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

Save the changes as `accordion7.html`. When you roll over one of the headings, they will still open as normal, but if you click on one of the headings, the URL specified as the header's `href` attribute will be followed.

With `navigation` enabled, the widget will check for a fragment identifier at the end of the URL when the page loads. If there is a fragment identifier, the accordion will open the drawer whose heading's `href` attribute matches the fragment. So, if the second heading is clicked in this example, and then the page is refreshed, the second drawer of the accordion will be opened automatically. Therefore, it is important to ensure that the `href` attributes for each accordion header is unique to avoid conflicts in this situation.

# Accordion methodology

The accordion includes a selection of methods that allow you to control and manipulate the behavior of the widget programmatically. Some of the methods are common to each component of the library, such as the `destroy` method, which is used by every widget. We'll look at each of these methods in turn.

# Destruction

One method provided by the accordion is the `destroy` method. This method removes the accordion widget and returns the underlying mark-up to its original state. We'll use the default properties associated with accordion instead of the ones we configured for the last few examples. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 8</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus.</div></div>
    </div>
    <button id="accordionKiller">Kill it!</button>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //turn specified element into an accordion
       $("#myAccordion").accordion();

      //attach click hander to button
      $("#accordionKiller").click(function() {

        //destroy the accordion
        $("#myAccordion").accordion("destroy");
       });
     });
   </script>
  </body>
</html>
```

The `<body>` of the page contains a new `<button>` element, which can be used to destroy the accordion. The final `<script>` block also contains a new anonymous function. We use the standard jQuery library's `click()` method to execute some code when the targeted `<button>` element is clicked.

We use the same `accordian()` constructor method to destroy it as we did to create it. But this time, we supply the string `"destroy"` as an argument. This causes the class names added by the library to be removed, the opening and closing behavior of the headers to no longer be effective, and all of the previously hidden content will be made visible.

Because we used an `ID` selector in our theme file to style the accordion container, this element will retain its size and borders. The roll-over effects were added by targeting the class names created by the library. As these are removed, along with the rest of the accordion's functionality, the rollovers do not activate. Save this file as `accordion8.html`.

# Enabling and disabling

Two very simple methods to use are `enable` and `disable`. These are just as easy to use as `destroy`, although they do have some subtle behavioral aspects that should be catered for in any implementation as you'll see. Change `accordion8.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 9</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus.</div></div>
    </div>
    <button id="enable">Enable!</button>
<button id="disable">Disable!</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
```

```
        //turn specified element into an accordion
        $("#myAccordion").accordion();

       //add click handler for enable button
       $("#enable").click(function() {

         //enable the accordion
         $("#myAccordion").accordion("enable");
       });

       //add click handler for disable button
       $("#disable").click(function() {

         //disable the accordion
         $("#myAccordion").accordion("disable");
       });
      });
    </script>
  </body>
</html>
```

We use these two methods in exactly the same way as the `destroy` method. Simply call `accordion()` with either `enable` or `disable` supplied as a string parameter. Save this file as `accordion9.html` and try it out.

One thing I'm sure you'll quickly notice is that when the accordion has been disabled, the rollover and selected effects are still apparent. This could be misleading as there is no visual cue that the widget has been disabled. This behavior is sure to be fixed in a later revision of the library. But for now, we can easily fix this with a little standard jQuery goodness and apply disabled states ourselves.

Another problem we have with our test page is that clicking the `Enable!` button while the accordion is already enabled does nothing. There is, of course, nothing for it to do. Some kind of indication that the widget is already enabled would be helpful. Let's see how easy it is to fix these minor issues. Update the current page to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 10</title>
  </head>
  <body>
    <div id="myAccordion">
```

```
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus.</div></div>
    </div>
    <button id="enable">Enable!</button>
<button id="disable">Disable!</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //turn specified element into an accordion
       $("#myAccordion").accordion().addClass("enabled");

      //add click handler for enable button
      $("#enable").click(function() {

         //alert if already enabled, enable and change classes if not
         ($("#myAccordion").hasClass("enabled")) ? alert("Accordion
already enabled!") : $("#myAccordion").accordion("enable").
addClass("enabled").removeClass("disabled") ;            });

      //add click handler for disable button
      $("#disable").click(function() {

        //alert if already disabled, disable and change classes if not
        ($("#myAccordion").hasClass("disabled")) ? alert("Accordion
already disabled!") : $("#myAccordion").accordion("disable").
addClass("disabled").removeClass("enabled") ;            });
     });
   </script>
  </body>
</html>
```

The new code takes care of notifying the visitor if they click the `Enable!` button while the accordion is already enabled, or if the `Disable!` button is clicked while it is already disabled, through simply adding two additional class names; `enabled` and `disabled`.

We use the standard jQuery `addClass()` method to initially set an additional class name of `enabled` on the accordion's container. A simple JavaScript `ternary` then looks for the presence of this class and invokes the `alert` if it is detected. This is done using the jQuery `hasClass()` method.

If the accordion is changed from enabled to disabled, the `addClass()`, and also the `removeClass()` methods are used to swap our class names appropriately. A less intrusive way for us to do this, without the need for alerts, would be to actually disable the `Enable!` button while the accordion is enabled and vice-versa. I'll leave you to try this on your own.

Save this as `accordion10.html`. Now we can add some new styles to our stylesheet to address our new `disabled` class. Open `accordionTheme.css` in your text editor and add the following new selectors and rules after the existing ones:

```
/* disabled state */
.disabled a {
  background:url(../img/accordion/disabled.gif) repeat-x 0px 0px;
  cursor:default;
}
.disabled a.selected {
  background:url(../img/accordion/disabled.gif) repeat-x 0px 0px;
  cursor:default;
}
.disabled a:hover {
  background:url(../img/accordion/disabled.gif) repeat-x 0px 0px;
  cursor:default;
}
```

Save this as `accordionTheme2.css` (don't forget to update the link to the stylesheet in the `<head>`). Now, when the `Disable!` button is clicked, the new class name will pick up our grayed out headings. As we've specified the same background image for the `selected` and `hover` states, the accordion will not appear to respond in any way to clicks or mouse overs while disabled.

# Drawer activation

The final method exposed by accordion is the `activate` method. This can be used to programmatically show or hide different drawers. We can easily test this method using a text box and a new button. Change `acordion10.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 11</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti. </div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus. </div></div>
    </div>
    <p>Choose a drawer to open</p>
    <input id="choice" type="text"><button id="activate">Activate
</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
```

```
        //turn specified element into an accordion
         $("#myAccordion").accordion();

        //add click handler for activate button
        $("#activate").click(function() {

          //get the value from the text box
          var choice = $("#choice").val();

          //open the chosen drawer
          $("#myAccordion").accordion("activate", choice - 1);

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

Save this file as `accordion11.html`. The `activate` method is used in the same way as the `destroy` method. It is passed to the `accordion()` constructor as an argument. Apart from supplying the string `"activate"`, we also need to tell the accordion which drawer to activate using a number representing the drawer's index.

Like standard JavaScript arrays, the index numbers for the accordion drawer headings begin with zero. Therefore, to open the correct drawer, we subtract 1 from the figure entered into the text box when we call the `activate` method.

# Accordion animation

You may have noticed the default slide animation built into the accordion. Apart from this, there are two other built-in animations that we can easily make use of. We can also switch off animations entirely by supplying `false` as the value of the `animated` property, although this doesn't look too good!

The other values we can supply are `bounceslide` and `easeslide`. However, these aren't actually unique animations as such. These are different easing styles which don't change the animation itself but instead, alter the way it runs. You should note at this stage that additional jQuery plugins are required for these easing methods.

For example, the `bounceslide` easing method causes the opening drawer to appear to bounce up and down slightly as it reaches the end of the animation. On the other hand, `easeslide` makes the animation begin slowly and then builds up to its normal speed. Let's take a moment to look at these different easing methods now. Change `accordion11.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme2.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 12</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div>Wow, look at all this content
that can be shown or hidden with a simple click!</div></div>
      <div><a href="#">Header 2</a><div>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean sollicitudin. Sed interdum
pulvinar justo. Nam iaculis volutpat ligula. Integer vitae felis quis
diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut posuere,
mauris at sodales rutrum, turpis tellus fermentum metus, ut bibendum
velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div>Donec at dolor ac metus
pharetra aliquam. Suspendisse purus. Fusce tempor ultrices libero. Sed
quis nunc. Pellentesque tincidunt viverra felis. Integer elit mauris,
egestas ultricies, gravida vitae, feugiat a, tellus. </div></div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery.easing.1.3.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery.easing.compatibility.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
```

```
        //set custom easing
        var accOpts = {
          animated: "bounceslide"
        }

        //turn specified element into an accordion
         $("#myAccordion").accordion(accOpts);

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

Save this file as `accordion12.html` We've used a couple of new script files in the source code. The `jquery.easing.1.3.js` file is the latest version of the easing plugin, and the `jquery.easing.compatibility.js` plugin which enables the latest version of the easing file to work without any further modifications. The easing type names were renamed in version 1.2 of the easing plugin. Both of these files are included in the downloadable code for this chapter, and they can also be found on the jQuery site.

The built-in easing effects, based on a series of equations created by Robert Penner in 2006, are very easy to use and create a great effect which can help build individuality into accordion implementations.



**Plugins**

There are many jQuery plugins available. These are often developed by the open-source community instead of the library's authors and can be used with jQuery and jQuery UI. A good place to find plugins is on the jQuery site itself at `http://plugins.jquery.com/`

Some of these plugins, such as the easing plugin, work with the library components, while other plugins, such as the compatibility plugin, assist other plugins. We will look at more plugins throughout the course of this book.

# Accordion events

The accordion defines the custom `change` event which is fired after a drawer on the accordion opens or closes. To react to this event, we can use the `change` configuration property to specify a function to be executed every time the event occurs. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
```

```
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
accordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Widget Example 13</title>
  </head>
  <body>
    <div id="myAccordion">
      <span class="corner topLeft"></span><span class="corner
topRight"></span><span class="corner bottomLeft"></span>
<span class="corner bottomRight"></span>
      <div><a href="#">Header 1</a><div id="panel1">Wow, look at all
this content that can be shown or hidden with a simple click!</div>
</div>
      <div><a href="#">Header 2</a><div id="panel2">Lorem ipsum dolor
sit amet, consectetuer adipiscing elit. Aenean sollicitudin. Sed
interdum pulvinar justo. Nam iaculis volutpat ligula. Integer vitae
felis quis diam laoreet ullamcorper. Etiam tincidunt est vitae est. Ut
posuere, mauris at sodales rutrum, turpis tellus fermentum metus, ut
bibendum velit enim eu lectus. Suspendisse potenti.</div></div>
      <div><a href="#">Header 3</a><div id="panel3">Donec at dolor
ac metus pharetra aliquam. Suspendisse purus. Fusce tempor ultrices
libero. Sed quis nunc. Pellentesque tincidunt viverra felis. Integer
elit mauris, egestas ultricies, gravida vitae, feugiat a, tellus.
</div></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

      //define config object
      var accOpts = {

      //add change event callback
          change: function(e, ui) {
            alert($(ui.newContent).attr("id") + " was opened, " +
$(ui.oldContent).attr("id") + " was closed");
          }
        };

        $("#myAccordion").accordion(accOpts);
      });
    </script>
  </body>
</html>
```

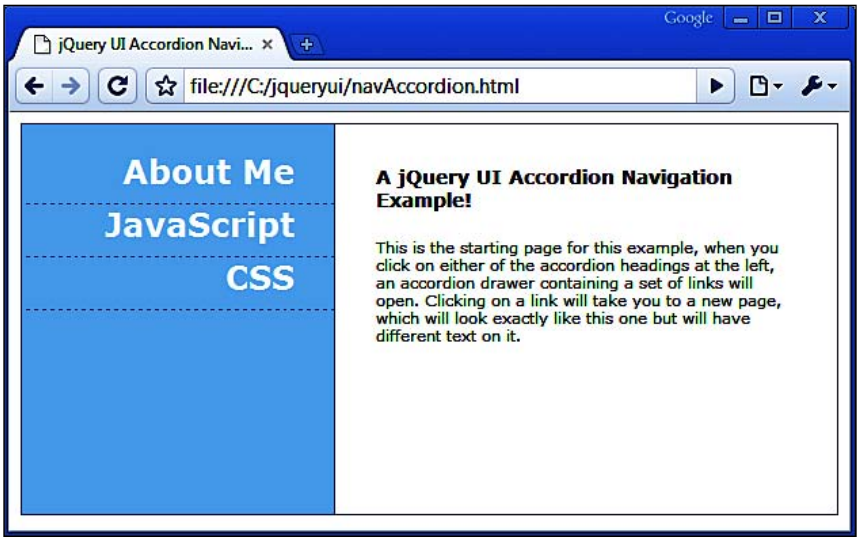
Save this as `accordion13.html`. In this example, we use the `change` configuration property to specify an anonymous callback function which is executed every time the event is triggered. This function will automatically receive two objects as arguments. The first object is the event object which contains information about the event. The second object is an object containing useful information about the accordion widget, such as the content drawer that just opened or closed.

In the mark-up for the accordion, we have given each of the content drawer `<div>` elements an `id` attribute which can be used in the alert generated by the change callback. We can use the `ui.newContent` and `ui.oldContent` properties to obtain the relevant content drawer and display its `id` in the alert.

The accordion widget also defines the accordion change event which is fired after a drawer on the accordion opens or closes. To react to this event, we can use the standard jQuery `bind()` method to specify a callback function, just like with the tabs widget from the last chapter.

# Fun with accordion

Let's put a sample together that will make the most of the accordion widget and uses some of the properties and methods that we've looked at so far in this chapter. A popular implementation of accordion is as a navigation menu. Let's build one of these based on the accordion widget. The following screenshot shows the finished page:



In a new page in your text editor, create the following HTML file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.accordion.css">
    <link rel="stylesheet" type="text/css" href="styles/
navAccordionTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Accordion Navigation Example</title>
  </head>
  <body>
      <div id="container">
        <div id="navCol">
          <ul id="navAccordion">
            <li>
              <a class="heading" href="#me" title="About Me">About
Me</a>
              <div>
                <a href="bio.html#me" title="Bio">My Bio</a>
                <a href="contact.html#me" title="Contact Me">Contact
Me</a>
                <a href="resume.html#me" title="Resume">My Resume</a>
              </div>
            </li>
            <li>
              <a class="heading" href="#js" title="JavaScript">
JavaScript</a>
              <div>
                <a href="tutorials.html#js" title="JavaScript
Tutorials">JavaScript Tutorials</a>
                <a href="ajax.html#js" title="AJAX">AJAX</a>
                <a href="apps.html#js" title="JavaScript
Apps">JavaScript Apps</a>
              </div>
            </li>
            <li>
              <a class="heading" href="#css" title="CSS">CSS</a>
              <div>
                <a href="layouts.html#css" title="Layouts">Layouts</a>
                <a href="themes.html#css" title="Themes">Themes</a>
                <a href="hacks.html#css" title="Hacks">Hacks</a>
```

```
              </div>
            </li>
          </ul>
        </div>
        <div id="contentCol">
          <h1>A jQuery UI Accordion Navigation Example!</h1>
          <p>This is the starting page for this example, when you
click on either of the accordion headings at the left, an accordion
drawer containing a set of links will open. Clicking on a link will
take you to a new page, which will look exactly like this one but will
have different text on it.</p>
        </div>
        <div id="clear"></div>
      </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.accordion.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

         //turn specified element into an accordion
       $("#navAccordion").accordion({
            header: ".heading",
            event: "mouseover",
            autoHeight: false,
            alwaysOpen: false,
            active:false,
            navigation: true
         });
      });
   </script>
  </body>
</html>
```

Save this as `navAccordion.html`. To see the full effects of the `navigation` property, the other pages that the main page links to should be available. Don't worry about creating these yourself, they are all in the code download for this chapter.

We use a selection of configurable properties in this example. The `header` property allows us to target only the links that have the class name heading. This prevents the links in the content sections from picking up any header attributes. We make use of the `event` property again to specify mouse over as the trigger event.

Switching off the `autoHeight` property prevents unnecessary whitespace in the menu from showing if there is one content section with much more content in it than other sections. The `alwaysOpen` property allows all headings to be closed. Disabling the `active` property also allows the page to load with all headings closed which is helpful if someone is visiting the application for the first time.

In order to make the most of the `navigation` property in this example, we make sure that each of the links that lead to new pages also include a fragment identifier matching the `href` of their heading element. Therefore, when a new page opens the state of the menu is maintained.

We'll also need some CSS for this example, just to make the page and the accordion look as we want them to. In a new file in your text editor, add the following stylesheet:

```
/* page */
#clear { clear:both; }
#container { border:1px solid #4e82b4; width:601px; }
#navCol {
  width:230px; height:287px; float:left; background-color:#a1d2f6;
}
#contentCol {
  width:310px; height:227px; float:left; background-color:#ffffff;
  padding:30px; border-left:1px solid #4e82b4;
}
h1 { margin:0px; font:bold 14px Verdana; }
#contentCol p { margin:20px 0 0 0; font:normal 11px Verdana; }

/* accordion  */
#navAccordion {
  list-style-type:none; padding-left:0; text-align:right;
  margin:20px 0 0 0; width:231px; position:relative; left:0;
}
#navAccordion a {
  display:block; text-decoration:none; font:bold 11px Verdana;
  color:#000000; padding:0 40px 0 0; padding-bottom:5px;
}
#navAccordion a:hover { text-decoration:underline; }
#navAccordion a.heading {
  font:bold 24px Verdana; color:#ffffff;
  border-bottom:1px dashed #4e82b4; padding:0 30px 10px 0;
}
#navAccordion a.heading:hover { text-decoration:none; }
.selected, #navAccordion .selected a.heading {
  background-color:#ffffff; color:#000000; border-top:0;
```

```
    border-bottom:1px solid #4e82b4; border-right:1px solid #ffffff;
    border-left:1px solid #ffffff;
}
#navAccordion .selected a.heading { border:0; }
#navAccordion li { margin:0; }
#navAccordion li span, #navAccordion li a { background-image:none; }
#navAccordion li span { display:none; }
```

Save this as `navAccordionTheme.css` in the `styles` folder. I've tried to keep the page and CSS code as minimal as possible, although a certain minimum amount of coding is going to be required in any practical example.

If you run `navAccordion.html` in your browser now, and then click on any of the links within each content section, you'll navigate to a new page. Thanks to the `navigation:true` name:value pair, the relevant section of the accordion will be open when the new page loads as seen below:




# Summary

The accordion widget allows us to easily implement an object on the page which will show and hide different blocks of content. This is a popular, and much sought after, effect which is implemented by big players on the web today like Apple.

We first saw that the accordion widget doesn't require any CSS at all in order to function as the behaviour without styling still works perfectly. We also looked at the `flora` styling, as well as the ease in which custom styles can be added.

We then moved on to look at the configurable properties that can be used with accordion. We saw that we can use these properties to change the behaviour of the widget, such as specifying an alternative heading to be open by default, whether the widget should expand to fill the height of its container, or the event that triggers the opening of a content drawer.

In addition to looking at these properties, we also saw that there are a range of methods which can be called on the accordion to make it do things programmatically. For example, we can easily specify a drawer to open, enable and disable any drawers, or even completely remove the widget and return the mark-up to its original state.

Finally, we looked at accordian's default animation and how we can add simple transition effects to the opening of content drawers. Like tabs, this is a flexible and robust widget that provides essential functionality and interaction in an aesthetically pleasing way.

# 4
# The Dialog

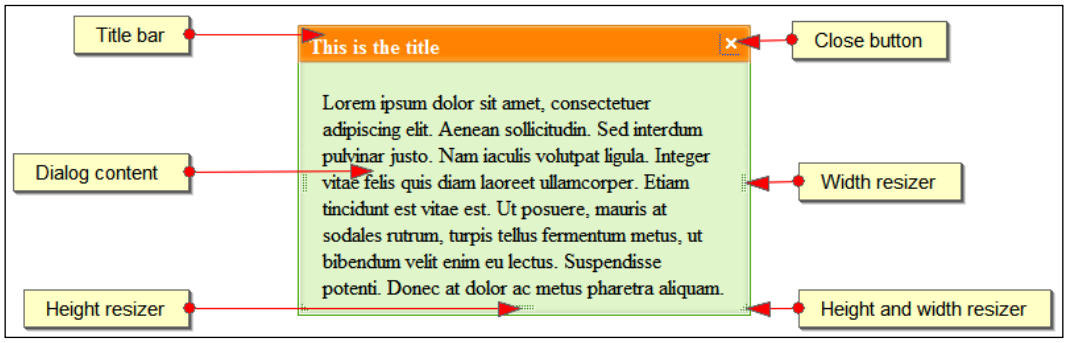
Traditionally, the way to display a brief message or ask a visitor a question would be to use one of JavaScript's native dialog boxes, such as `alert` or `confirm`, or to open a new web page with a predefined size, styled to look like a dialog box.

Unfortunately, as I'm sure you're aware, neither of these methods is particularly flexible or engaging. For each problem they solve, several new problems are usually introduced.

Thankfully, the days of resorting to either of the aforementioned techniques are over. We can now make use of the advanced functionality and rich features of the jQuery UI dialog widget.

The dialog widget lets us display a message, supplemental content (like images or text), or even interactive content (like forms). It's also very easy to add buttons, such as simple **ok** and **cancel** buttons, to the dialog and define callback functions for them in order to react to their being clicked.

The following screenshot shows a dialog widget and the different elements that it is made of:



In this chapter, we will complete the following tasks:

- Create a basic dialog
- Create a custom dialog skin
- Work with dialog's properties
- Enable modality and see an overlay
- Add buttons to the dialog
- Work with dialog's callbacks
- Enable animations for the dialog
- Control the dialog programmatically

# A basic dialog

A dialog has a lot of default behavior built-in, but few methods are needed to control it programmatically, making this a very easy widget to use that is also highly configurable.

Generating it on the page is very simple and requires a minimal underlying mark-up structure. The following page contains the minimum mark-up that's required to implement the dialog widget:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 1</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">Lorem
    ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
    sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
    ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
    tincidunt est vitae est. Ut posuere, mauris at sodales rutrum,
    turpis tellus fermentum metus, ut bibendum velit enim eu lectus.
    Suspendisse potenti. Donec at dolor ac metus pharetra aliquam.
    Suspendisse purus. Fusce tempor ultrices libero. Sed quis nunc.
```

```
    Pellentesque tincidunt viverra felis. Integer elit mauris,
    egestas ultricies, gravida vitae, feugiat a, tellus.</div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create the dialog
        $("#myDialog").dialog();

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

Save this as `dialog1.html` in the `jqueryui` folder. A few more source files are required that we haven't used before, specifically the `ui.resizable.js` and `ui.draggable.js` files and the `flora.resizable.css` stylesheet.

The JavaScript files are low-level interaction helpers, which we'll be covering in more detail later on in the book, and are only required if the dialog is going to be resizable and draggable. The widget will still function without them. The dialog `flora` theme file is a mandatory requirement for this component, although the resizable one isn't.

Other than that, the widget is initialized in the same way as other widgets we have already looked at. When you run this page in your browser, you should see the default dialog widget shown in the previous screenshot, complete with draggable and resizable behaviors.

One more feature that I think deserves mentioning here is modality. The dialog comes with modality built-in, although it is disabled by default. When modality is enabled, a modal overlay element, which covers the underlying page, will be applied. The dialog will sit above the overlay while the rest of the page will be below it.

The benefit of this feature is that it ensures the dialog is closed before the underlying page becomes interactive again, and gives a clear visual indicator that the dialog must be closed before the visitor can proceed.

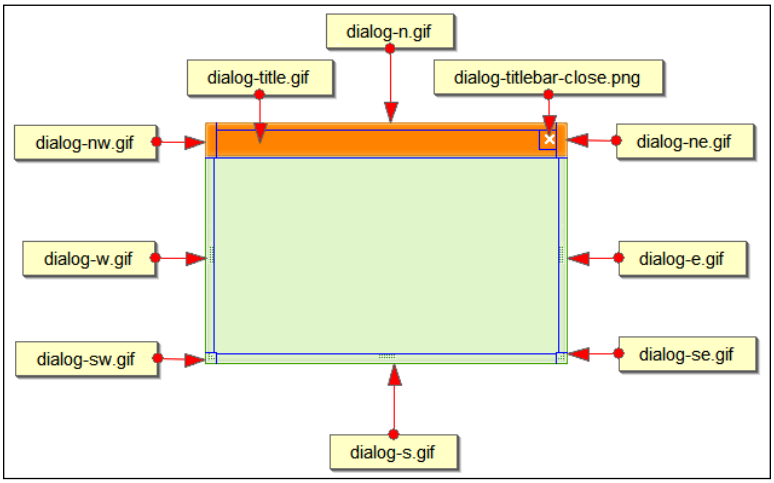
# Custom dialog skins

The dialog's appearance is easy to change from the `flora` theme used in the first example. Like some of the other widgets we have looked at, certain aspects of the `default` or `flora` themes are required to make the widget function correctly. Therefore, when overriding styles, we need to be careful to just override the rules related to the dialog's display.

When creating a new skin for the default implementation, including resizable behavior, we have a lot of new files that will need to be created. Apart from new images for the different components of the dialog, we also have to create new images for the resizing handles. The following files need to be replaced when skinning a dialog:

- `dialog-e.gif`
- `dialog-n.gif`
- `dialog-ne.gif`
- `dialog-nw.gif`
- `dialog-s.gif`
- `dialog-se.gif`
- `dialog-sw.gif`
- `dialog-title.gif`
- `dialog-titlebar-close.png`
- `dialog-titlebar-close-hover.png`

To make it easier to remember which image corresponds to which part of the dialog, these images are named after the compass points at which they appear. The following image illustrates this:



Note that these are file names as opposed to class names. The class names given to each of the different elements that make up the dialog, including resizable elements, are similar, but are prefixed with `ui-` as we'll see in the next example code.

Let's replace these images with some of our own (the necessary files can be found in the code download). In a new file in your text editor, create the following stylesheet:

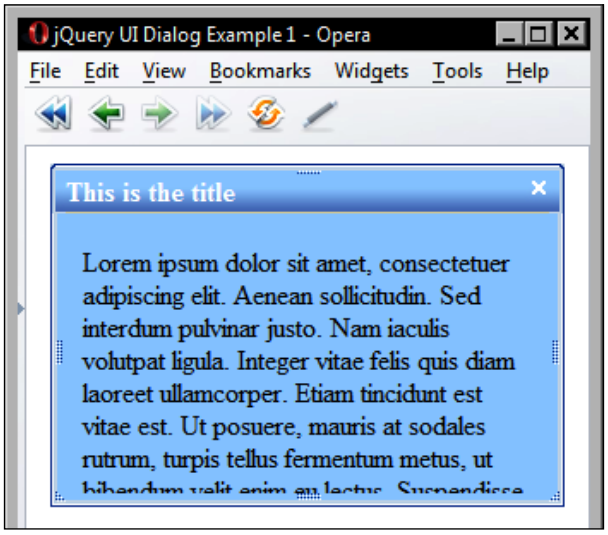
```
.flora .ui-dialog, .flora.ui-dialog {
  background-color:#99ccff;
}
.flora .ui-dialog .ui-dialog-titlebar, .flora.ui-dialog
.ui-dialog-titlebar {
  background:url(../img/dialog/my-title.gif) repeat-x;
  background-color:#003399;
}
.flora .ui-dialog .ui-dialog-titlebar-close, .flora.ui-dialog
.ui-dialog-titlebar-close {
  background:url(../img/dialog/my-title-close.gif) no-repeat; }
.flora .ui-dialog .ui-dialog-titlebar-close-hover, .flora.ui-dialog
.ui-dialog-titlebar-close-hover {
  background:url(../img/dialog/my-title-close-hover.gif) no-
  repeat;
}
.flora .ui-dialog .ui-resizable-n, .flora.ui-dialog .ui-resizable-n {
  background:url(../img/dialog/my-n.gif) repeat center top;
}
.flora .ui-dialog .ui-resizable-s, .flora.ui-dialog .ui-resizable-s {
  background:url(../img/dialog/my-s.gif) repeat center top;
}
.flora .ui-dialog .ui-resizable-e, .flora.ui-dialog .ui-resizable-e {
  background:url(../img/dialog/my-e.gif) repeat right center; }
.flora .ui-dialog .ui-resizable-w, .flora.ui-dialog .ui-resizable-w {
  background:url(../img/dialog/my-w.gif) repeat left center;
}
.flora .ui-dialog .ui-resizable-ne, .flora.ui-dialog .ui-resizable-ne
{
  background:url(../img/dialog/my-ne.gif) repeat;
}
.flora .ui-dialog .ui-resizable-se, .flora.ui-dialog .ui-resizable-se
{
  background:url(../img/dialog/my-se.gif) repeat;
}
.flora .ui-dialog .ui-resizable-sw, .flora.ui-dialog .ui-resizable-sw
{
  background:url(../img/dialog/my-sw.gif) repeat;
}
.flora .ui-dialog .ui-resizable-nw, .flora.ui-dialog .ui-resizable-nw
{
  background:url(../img/dialog/my-nw.gif) repeat;
}
```

Save this as `dialogTheme.css` in the `styles` folder. We should also create a new folder within our `img` folder called `dialog`. This folder will be used to store all of our dialog-specific images.

All we need to do is specify new images to replace the existing ones used by `flora`. All other rules can stay the same. In `dialog1.html`, link to the new file with the following code, which should appear directly after the link to the resizable stylesheet:

```
<link rel="stylesheet" type="text/css" href="styles/dialogTheme.css">
```

Save the change as `dialog2.html`. These changes will result in a dialog that should appear similar to the following screenshot:




So you can see that skinning the dialog to make it fit in with your existing content is very easy. The existing image files used by the default theme give you something to start with, and it's really just a case of playing around with colors in an image editor until you get the desired effect.

# Dialog properties

An options object can be used in a dialog's constructor method to configure various dialog properties. Let's look at the available properties:

| Property | Default Value | Usage |
|---|---|---|
| `autoOpen` | `true` | Shows the dialog as soon as the `dialog` method is called |
| `bgiframe` | `true` | Creates an `<iframe>` shim to prevent `<select>` elements showing through the dialog in IE6 - at present, the `bgiframe` plugin is required, although this may not be the case in future versions of this widget |
| `buttons` | {} | Supplies an object containing buttons to be used with the dialog |
| `dialogClass` | `ui-dialog` | Sets additional class names on the dialog for theming purposes |
| `draggable` | `true` | Makes the dialog draggable (use `ui.draggable.js`) |
| `height` | `200`(px) | Sets the starting `height` of the dialog |
| `hide` | *none* | Sets an effect to be used when the dialog is closed |
| `maxHeight` | *none* | Sets a maximum `height` for the dialog |
| `maxWidth` | *none* | Sets a maximum `width` for the dialog |
| `minHeight` | `100`(px) | Sets a minimum `height` for the dialog |
| `minWidth` | `150`(px) | Sets a minimum `width` for the dialog |
| `modal` | `false` | Enables modality while the dialog is open |
| `overlay` | {} | Object with CSS properties for the modal overlay |
| `position` | `center` | Sets the starting position of the dialog in the viewport |
| `resizable` | `true` | Makes the dialog resizable (also requires `ui.resizable.js`) |
| `show` | *none* | Sets an effect to be used when the dialog is opened |
| `stack` | `true` | Causes the focused dialog to move to the front when several dialogs are open |
| `title` | *none* | Alternative to specifying `title` on source element |
| `width` | `300`(px) | Sets the original width of the dialog |

As you can see, we have a range of configurable properties to work with in our dialog implementations. Many of these properties are boolean or numerical, and string-based, making them extremely easy to set and work with.

In our examples so far, the dialog has opened as soon as the page has loaded, or as soon as the `dialog` constructor method is called, which is as soon as the page is ready in this case. We can change this so that the dialog is opened when something else occurs. For example, a `<button>` being clicked, say, by adjusting the `autoOpen` property. We'll come back to this property when we look at the `open` method a little later on.

The `position` property controls where the dialog is rendered in the viewport when it is opened and accepts either a string or an array value. The strings may be one of the following values:

- `bottom`
- `center`
- `left`
- `right`
- `top`

An array is used when you want to specify the exact coordinates of the top-left corner where the dialog should appear. The coordinates are specified as an offset from the top-left corner of the viewport.

The previous table shows a `title` property. Although the title for the dialog can be set using the `title` attribute of the underlying HTML element, using the `title` property is preferred. This will stop the dialog from displaying the `title` when the body of the dialog widget is hovered over.

One of the dialog's greatest assets is modality. This feature creates an overlay when the widget is opened that sits above the page content but below the dialog. The overlay is removed as soon as the dialog is closed. But while the dialog is open, none of the underlying page content can be interacted with in any way.

When using the modal feature of the dialog widget, we also need to configure the `overlay` property too. Change `dialog2.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 3</title>
  </head>
  <body>
```

```
    <div id="myDialog" class="flora" title="This is the title">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //define config object
        var dialogOpts = {
          modal: true,
          overlay: {
            background: "url(img/modal.png) repeat"
          }
        };
        //create the dialog
        $("#myDialog").dialog(dialogOpts);
      });
    </script>
  </body>
</html>
```

This file can be saved as `dialog3.html`. When you view the page in a browser, you'll see the modal effect immediately.

We've used a repeated, semi-transparent PNG image for the overlay in this example for simplicity, but other CSS properties such as background colors and opacity are also acceptable. The path to the PNG is specified as the value of the `overlay` property, so you can see how this property and the `modal` property should be used together.

The PNG used in this example is a simple square one pixel high by one pixel wide. The image is made of the color `#999999` set to approx 25% transparency. This file is included with the code download, but is also very easy to make.

# Adding buttons

One of the properties we saw a moment ago was the `button` property. This property accepts a literal object and is used to specify the `<button>` elements that should be present on the dialog. Let's add an **ok!** `<button>` to our dialog. Alter `dialog3.html` so that it appears like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>

    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 4</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus.</div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
```

```
        //define doOk function
        var doOk = function() {
        }
        //define config object
        var dialogOpts = {
          modal: true,
          overlay: {
            background: "url(img/modal.png) repeat",
          },
          buttons: {
            "Ok!": doOk
          },
          height: "250px"
        };
        //create the dialog
        $("#myDialog").dialog(dialogOpts);
      });
    </script>
  </body>
</html>
```

Save the file as `dialog4.html`. The key for each property in the `buttons` object is the text that will form the `<button>` label, and the value is the name of the callback function to execute when the `<button>` is clicked.

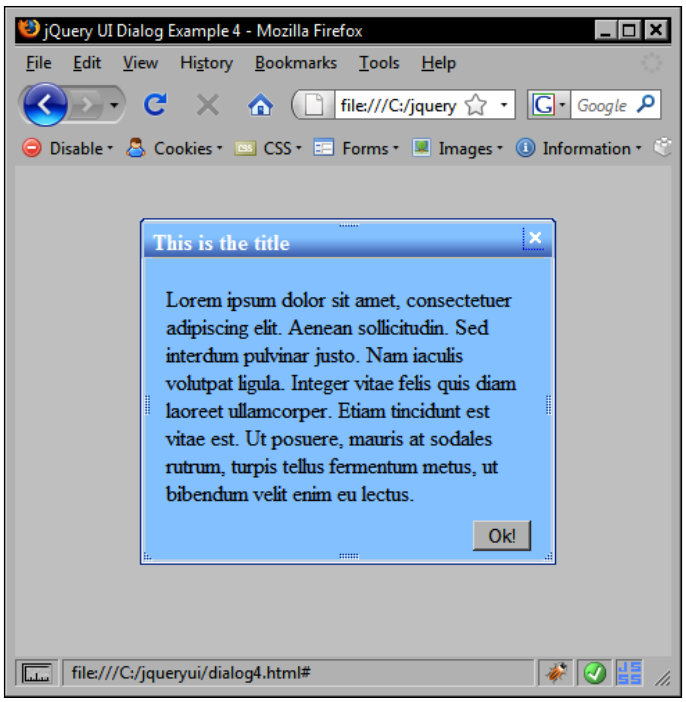
We've added the `doOk` function as the behavior for our new `<button>`. Although it won't do anything at this stage, the page won't work without it. We can come back to this function in a little while when we look at a dialog's methods.

We also configured the `height` property in this example. The buttons that we create are absolutely positioned at the bottom of the dialog. Therefore, we need to increase the height of the widget so that the `<button>` does not obscure the text.

We can also style the `<button>` itself by using the following selector. We need to do this in order to move the `<button>` up from the bottom edge of the widget slightly. Add the following code to `dialogTheme.css`:

```
.flora .ui-dialog .ui-dialog-buttonpane button, .flora.ui-dialog
.ui-dialog-buttonpane button { margin-bottom:10px; }
```

View the new file in your browser. It should appear something like the following screenshot:




# Working with dialog's callbacks

The dialog widget gives us a wide range of callback properties that we can use to execute arbitrary code at different points in any dialog interaction. The following table lists the properties available to us:

| Property | Fired When |
|---|---|
| `close` | The dialog is closed |
| `drag` | The dialog is being dragged |
| `dragStart` | The dialog starts being dragged |
| `dragStop` | The dialog stops being dragged |
| `focus` | The dialog receives focus |
| `open` | The dialog is opened |
| `resize` | The dialog is being resized |
| `resizeStart` | The dialog starts to be resized |
| `resizeStop` | The dialog stops being resized |

Some of these callbacks are only available in certain situations, such as the `drag` and `resize` callbacks, while others such as the `open`, `close`, and `focus` callbacks, will be available in any implementation. Let's look at an example in which we can make use of some of these callback properties. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 5</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</div>
    <p id="status">The dialog is closed</p>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //define config object
```

```
        var dialogOpts = {
          open: function() {
            //change status
            $("#status").text("The dialog is open");
          },
          close: function() {
            //change status
            $("#status").text("The dialog is closed");
          }
        };
        //create the dialog
        $("#myDialog").dialog(dialogOpts);

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

Save this as `dialog5.html`. Our configuration object uses the `open` and `close` properties to specify simple callback functions. These change the text of the status message depending on the state of the dialog.

When the dialog is opened, the text will be changed to reflect this, and likewise, when it is closed, the text will be changed. It's a simple page, but it highlights the points at which the `open` and `close` events are fired and shows how easy these properties are to use.

# Using dialog animations

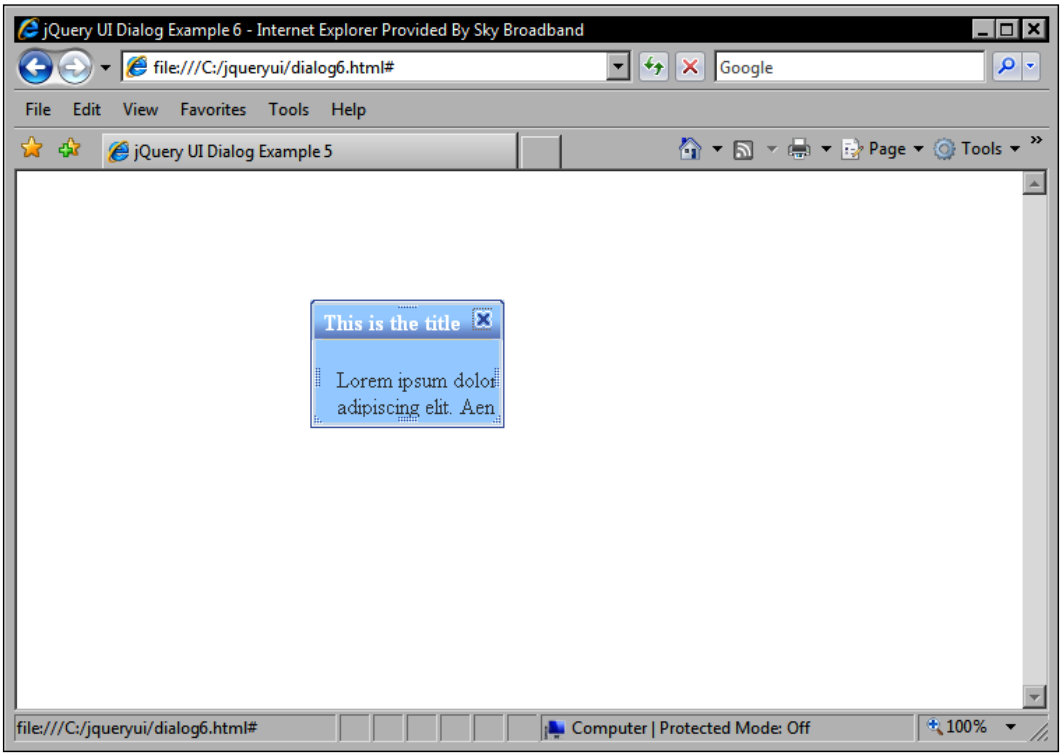
The dialog provides us with built-in effect abilities and also allows us to specify effects to use when the dialog is opened or closed. Using these effects is extremely easy and gives a great visual flair. Let's look at how these effects can be enabled. Create the following new page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 6</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

       //define config object
       var dialogOpts = {
         hide: true
       };

        //create the dialog
        $("#myDialog").dialog(dialogOpts);
      });
    </script>
  </body>
</html>
```

Save this as `dialog6.html`. In this example, our configuration object contains just one property—the `hide` property. The `hide` property accepts the boolean `true` as its value. This enables the built-in `hide` effect, which gradually reduces the dialog's size and opacity until it gracefully disappears.

We can also enable the `show` effect, which is the opposite of the `hide` animation. However, at this stage in the component's development, this causes a slight issue with its display. The following screenshot shows the `hide` effect in progress:




# Controlling a dialog programmatically

The dialog requires few methods in order to function. As implementers, we can easily open, close, or destroy the dialog at will. The full list of methods we can call on a dialog instance are as follows:

| Method | Used to |
|---|---|
| `close` | Closes or hides the dialog |
| `destroy` | Permanently disables the dialog |
| `isOpen` | Determines whether a dialog is open or not |
| `moveToTop` | Moves the specified dialog to the top of the stack |
| `open` | Opens the dialog |

Let's look at opening and closing the widget, which can be achieved with the simple use of the `open` and `close` methods. Create the following new page in your text editor:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>

    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 7</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</div>
    <button id="openDialog">Open the Dialog!</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //define doOk function
        var doOk = function() {
```

```
            //close the dialog
            $("#myDialog").dialog("close");
          }
          //define config object
          var dialogOpts = {
            modal: true,
            overlay: {
              background: "url(img/modal.png) repeat"
            },
            buttons: {
              "Ok!": doOk
            },
            height: "400px",
            autoOpen: false
          };
          //create the dialog
          $("#myDialog").dialog(dialogOpts);
          //define click handler for the button
          $("#openDialog").click(function() {
            //open the dialog
            $("#myDialog").dialog("open");
          });
        });
      </script>
    </body>
  </html>
```

The `open` and `close` methods require no additional arguments and do exactly as they say, pure and simple. Save the file as `dialog7.html`.

The `destroy` method for a dialog works in a slightly different way than it does for the other widgets we've seen so far. Instead of returning the underlying HTML to its original state, the dialog's `destroy` method completely disables the widget, hiding its content in the process. Change `dialog7.html` to make use of the `destroy` method:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 8</title>
  </head>
  <body>
    <div id="myDialog" class="flora" title="This is the title">
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</div>
    <button id="openDialog">Open the Dialog!</button>
    <button id="destroy">Destroy the dialog!</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //define doOk function
        var doOk = function() {

          //close the dialog
          $("#myDialog").dialog("close");

        }

        //define config object
        var dialogOpts = {
          modal: true,
          overlay: {
            background:"url(img/modal.png) repeat"
```

```
            },
            buttons: {
             "Ok!": doOk
            },
            height: "400px",
            autoOpen: false
          };

          //create the dialog
          $("#myDialog").dialog(dialogOpts);

          //define click handler for the button
          $("#openDialog").click(function() {

            //open the dialog
            $("#myDialog").dialog("open");

          });

          //define click handler for destroy
          $("#destroy").click(function() {

            //destroy dialog
            $("#myDialog").dialog("destroy");

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

Save the changes as `dialog8.html` and try out the new file. You'll find that you can open and close the dialog as many times as you want until the destroy button is clicked. After this, the dialog will no longer appear (although it will still exist in the DOM). To fully remove the dialog mark-up from the page, we can simply chain the `remove` jQuery method onto the end of the `destroy` method call.

# Getting data from the dialog

Because the widget is a part of the underlying page, passing data to and from it is extremely simple. The dialog can be treated as any other standard element on the page. Let's look at a basic example. Create the following new page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Dialog Example 9</title>
  </head>
  <body>
    <p>Answer the opinion poll:</p>
    <button id="poll">Poll</button>
    <div id="myDialog" class="flora" title="This is the title">
      <p>Is jQuery UI the greatest JavaScript extensions library in
the universe?</p>
      <label for="yes">Yes!</label><input type="radio" id="yes"
value="yes" name="question"><br>
      <label for="no">No!</label><input type="radio" id="no"
value="no" name="question">
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //define cancel button function
        var cancel = function() {

          //close the dialog
          $("#myDialog").dialog("close");

        }

        //define done button function
        var getResponse = function(){

          var answer;
          $("input").each(function(){
```

```
            (this.checked == true) ? answer = $(this).val() : null;
          });
          $("<p>").text("Thanks for selecting " + answer)
.insertAfter($("#poll"));
          $("#myDialog").dialog("close");
        }
        //define config object
        var dialogOpts = {
          modal: true,
          overlay: {
            background: "url(img/modal.png) repeat"
          },
          buttons: {
            "Done": getResponse,
            "Cancel": cancel
          },
          autoOpen: false
        };
        //create the dialog
        $("#myDialog").dialog(dialogOpts);
        //define click handler for poll button
        $("#poll").click(function() {
          //open the dialog
          $("#myDialog").dialog("open");
        });
      });
    </script>
  </body>
</html>
```

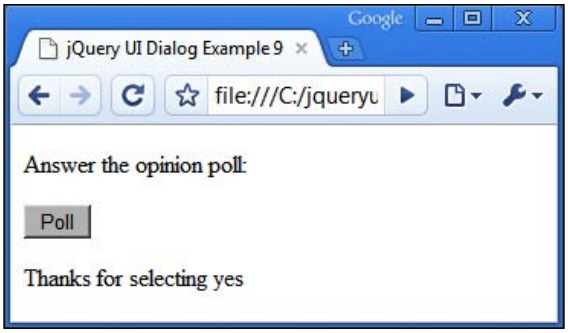
Save this as `dialog9.html`. Our dialog contains a set of radio buttons, `<label>` elements, and some text. The purpose of the example is to get the result of which radio is selected, and then do something with it when the dialog closes.

We start the `<script>` off by creating the `cancel` function, which will be attached as the value of the `cancel` property in the `buttons` object later in the script. It will therefore be executed each time the **Cancel** `<button>` is clicked.

Next, we define the `getResponse` function, which again will be attached to a `<button>` on the dialog using the `buttons` configuration object. In this function, we determine which radio is selected, then create and append to the page a new `<p>` element with a brief message and the value of the radio that was selected.

Once these two functions have been defined, we create a configuration object as before. The dialog is initially hidden from view, and we use the `open` method to show the dialog when the **Poll** `<button>` is clicked.

The following screenshot shows how the page should appear once a radio button has been selected:




# Fun with dialog

The class behind the dialog widget is compact and is catered to a small range of specialised behavior, much of which we have already looked at. We can still have some fun with the dialog widget however, and could, for example, easily create an AJAX dialog which gets its content from a remote file when it is opened. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>

    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.dialog.css">
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
ajaxDialogTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI AJAX Dialog Example</title>
```

```
  </head>
  <body>
    <div id="ajaxDialog" class="flora"></div>
    <div class="content">
      <h3>Section 1</h3>
      <p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aenean sollicitudin. Sed interdum pulvinar justo. Nam iaculis volutpat
ligula. Integer vitae felis quis diam laoreet ullamcorper. Etiam
tincidunt est vitae est. Ut posuere, mauris at sodales rutrum, turpis
tellus fermentum metus, ut bibendum velit enim eu lectus. Suspendisse
potenti. Donec at dolor ac metus pharetra aliquam. Suspendisse purus.
Fusce tempor ultrices libero. Sed quis nunc. Pellentesque tincidunt
viverra felis. Integer elit mauris, egestas ultricies, gravida vitae,
feugiat a, tellus.</p>
      <p class="helpLabel">For help about this section, click here:
</p><span id="help1" class="helpIcon"></span>
    </div>
  <div class="content">
    <h3>Section 2</h3>
    <p>Lorem ipsum...</p>
    <p class="helpLabel">For help about this section, click here:
</p><span id="help2" class="helpIcon"></span>
  </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.dialog.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
  </body>
</html>
```

Now add the following `<script>` block directly before the closing `</body>` tag:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){

    //control variable
    var help = 0;

    //define doOk function
    var doOk = function() {
```

```
        //close the dialog
        $("#ajaxDialog").dialog("close");
      }
      //define config object
      var dialogOpts = {
        title: "Help!",
        modal: true,
        overlay: {
          background: "url(img/modal.png) repeat"
        },
        buttons: {
          "Ok!": doOk
        },
        height: "110px",
        autoOpen: false,
        open: function() {
          //display correct dialog content
          $("#ajaxDialog").load("helpContents" + (help == 1 ? 1 : 2) +
".html");    }
      };
      //create the dialog
      $("#ajaxDialog").dialog(dialogOpts);
      //define click handler for helpIcons
      $(".helpIcon").click(function(event) {
        //which icons was clicked?
        event.target.id == "help1" ? help = 1 : help = 2;
        //call open method
        $("#ajaxDialog").dialog("open");
      });
    });
  </script>
```

Save this as `ajaxDialog.html`. The dialog is similar to that of previous examples, but the main differences are the `open` event handler defined within the dialog's configuration object, and the `click` handler for the `helpIcon` elements.

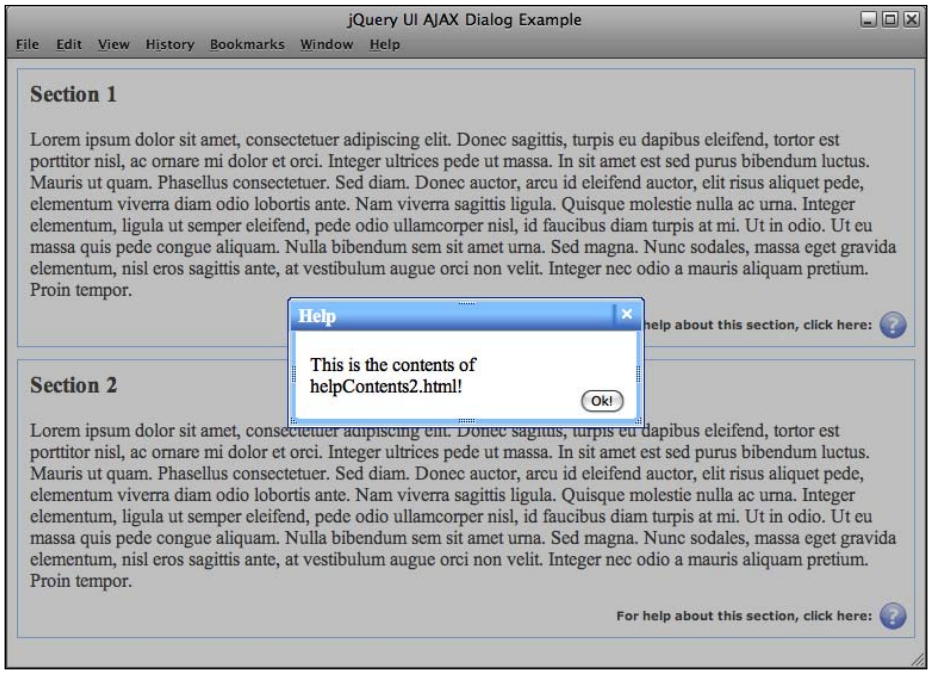
When either of the `helpIcon` elements are clicked, the handler will determine which icon it was, and set our control variable accordingly. The `open` method of the dialog is then called.

This will invoke the `open` event handler which reads the control variable and then loads the appropriate external file into the dialog using the standard jQuery `load` method. We'll need a new stylesheet for this example. In a new page in your text editor, add the following code:

```
* page styles */
h3 { margin-top:0px; }
.content {
  border:1px solid #7eb8f8;
  margin-bottom:10px; padding:10px;
  position:relative;
}
.helpIcon {
  position:absolute; right:5px; bottom:5px;
  background:url(../img/QuestionMark.png) no-repeat;
  cursor:pointer;
  display:block; width:25px; height:25px;
}
.helpLabel {
  width:100%; margin:0px;
  position:relative; right:25px; top:-2px;
  font:bold 60% Verdana, Arial; text-align:right;
}
/* dialog styles */
.flora .ui-dialog, .flora.ui-dialog {
  background-color:#ffffff;
}
.flora .ui-dialog .ui-dialog-titlebar, .flora.ui-dialog
.ui-dialog-titlebar {
  background:url(../img/dialog/my-title.gif) repeat-x;
background-color:#003399;
}
.flora .ui-dialog .ui-dialog-titlebar-close, .flora.ui-dialog
.ui-dialog-titlebar-close {
  background:url(../img/dialog/my-title-close.gif) no-repeat;
}
.flora .ui-dialog .ui-dialog-titlebar-close-hover, .flora.ui-dialog
.ui-dialog-titlebar-close-hover {
  background:url(../img/dialog/my-title-close-hover.gif) no-repeat;
}
.flora .ui-dialog .ui-resizable-n, .flora.ui-dialog .ui-resizable-n {
  background:url(../img/dialog/my-n.gif) repeat center top;
}
.flora .ui-dialog .ui-resizable-s, .flora.ui-dialog .ui-resizable-s {
  background:url(../img/dialog/my-s.gif) repeat center top;
}
.flora .ui-dialog .ui-resizable-e, .flora.ui-dialog .ui-resizable-e {
  background:url(../img/dialog/my-e.gif) repeat right center;
```

```
}
.flora .ui-dialog .ui-resizable-w, .flora.ui-dialog .ui-resizable-w {
  background:url(../img/dialog/my-w.gif) repeat left center;
}
.flora .ui-dialog .ui-resizable-ne, .flora.ui-dialog .ui-resizable-ne
{
  background:url(../img/dialog/my-ne.gif) repeat;
}
.flora .ui-dialog .ui-resizable-se, .flora.ui-dialog .ui-resizable-se
{
  background:url(../img/dialog/my-se.gif) repeat;
}
.flora .ui-dialog .ui-resizable-sw, .flora.ui-dialog .ui-resizable-sw
{
  background:url(../img/dialog/my-sw.gif) repeat;
}
.flora .ui-dialog .ui-resizable-nw, .flora.ui-dialog .ui-resizable-nw
{
  background:url(../img/dialog/my-nw.gif) repeat;
}
```

Many of these styles have been used in previous examples, but adding some new rules for the other page elements lets us see the dialog in real-world context. Save this as `ajaxDialogTheme.css` in the `styles` folder. Open the page and click the help icon in the second section. The dialog, with its correct content, should be displayed:





**Help Icon**

The icons used as help icons in this example were taken from the ColorCons icon package by Ken Saunders, and can be found at `http://mouserunner.com/Spheres_ColoCons1_Free_Icons.html`.

# Summary

The dialog widget is extremely specialized and is catered to the display of a message or question in a floating panel that sits above the page content. Advanced functionality, such as draggability and resizability, are directly built in, and features such as the excellent modality and overlay are easy to configure.

We started out by looking at the default implementation, which is as simple as it is with the other widgets we have looked at so far. However, there are several optional components that can also be used in conjunction with the dialog, such as the draggables and resizable components.

We then moved on to look at the different styling options available for use with the dialog, including the `default` or `flora` themes, and how easy it is to override some of these styles to create our own custom theme.

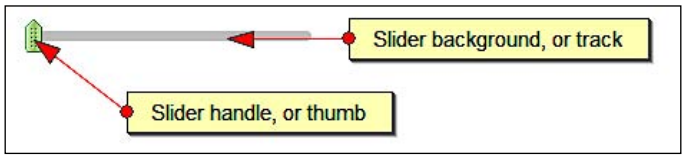
We also examined the range of configurable properties exposed by the dialog's API. We can easily make use of the properties to enable or disable built-in behavior such as modality, or set the dimensions of the widget, as well as giving us a wide range of callbacks that allow us to hook into custom events fired by the widget during an interaction.

We then took a brief look at the built-in opening and closing effects that can be used with the dialog, before moving on to see the basic methods we can invoke in order to make the dialog do things, such as open or close.

# 5
# Slider

The slider component allows us to implement an engaging and easy-to-use widget that our visitors should find attractive and intuitive to use. Its basic function is simple. The slider background represents a series of values which are selected by dragging the thumb along the background.

Before we roll up our sleeves and begin creating a slider, let's look at the different elements that it is made from. The following is an example of a slider:




It's a simple widget, as you can see, comprised of just two main elements. The slider handle, also called the thumb, and the slider background, also called the track. The only HTML elements created by the control are an `<a>` tag with a `<div>` element inside it, nothing else is dynamically generated.

In this section, we will cover the following topics:

- The default slider implementation
- Giving the slider a new appearance
- Creating a vertical slider
- Working with slider properties
- The slider's built-in event callbacks
- Making things happen with slider methods
- Sliders with multiple handles
- Working with slider ranges

# Implementing slider

Creating the default, basic slider takes no more code than any of the other widgets we have looked at so far. The underlying HTML mark-up required is also minimal. Let's create a basic one now. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 1</title>
  </head>
  <body>
    <div id="mySlider"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create the slider
        $("#mySlider").slider();

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

Save this file as `slider1.html` and view it in your browser. You should see something similar to the previous screenshot. We've used several library resources here, including the following files:

- `flora.slider.css`
- `jquery-1.6.2.js`
- `ui.core.js`
- `ui.slider.js`

Our container element is automatically given the class name `ui-slider`. This class name is targeted by the skin file and provides the background image that makes up the slider background, or track, as well as its positional and dimensional properties.

The default behavior of a basic slider is simple but effective. The thumb can be moved horizontally along any pixel of the track on the x axis, making allowances for the buffer and thumb width of course.

Clicking anywhere on the track, with the left or right mouse button, will instantly move the handle to that position. Once the handle has been selected, it is also possible to move it using the left and right arrow keys of the keyboard.

# Overriding the default theme

Altering the appearance of the slider is as easy as overriding the selectors that target the slider background and handle. To give the slider a completely different appearance, create the following page:

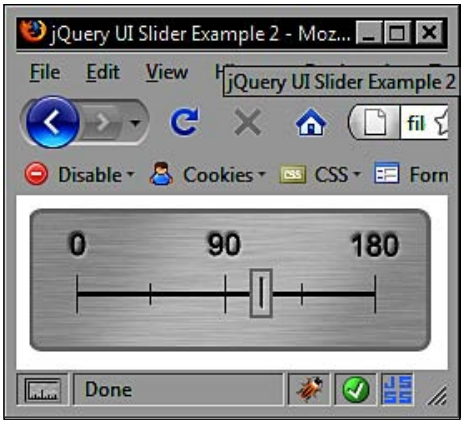
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <link rel="stylesheet" type="text/css" href="styles/
sliderTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 2</title>
  </head>
  <body>
    <div class="background-div">
      <div id="mySlider"></div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create the slider
```

```
        $("#mySlider").slider();

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

Save this as `slider2.html`. The only difference is that we've linked to a custom stylesheet. We also enclosed the slider within a container `<div>` which will be used for an additional background image to complete the appearance of our new slider. Now for that stylesheet, in a new file, add the code found below:

```
.background-div {
  background:url(../img/slider/slider_outerbg.gif) no-repeat;
  height:50px; width:217px; padding:36px 0 0 24px;
}
.ui-slider, .ui-slider-1 {
  background:url(../img/slider/slider_bg.gif) no-repeat; width:184px;
  height:23px; position:relative; left:4px; top:4px;
}
.ui-slider-handle {
  background:url(../img/slider/slider_handle.gif) no-repeat;
  width:14px; height:30px; top:-4px;
}
```

Save the file as `sliderTheme.css` in your `styles` folder. Create a new folder inside the `img` folder and name it `slider`. You should put the images from the code download for this chapter into the new folder. Make sure you link to the new stylesheet in `slider1.html` and save the new file as `slider2.html`. When you view the new file in your browser, the slider should look completely different, as shown in the following screenshot:



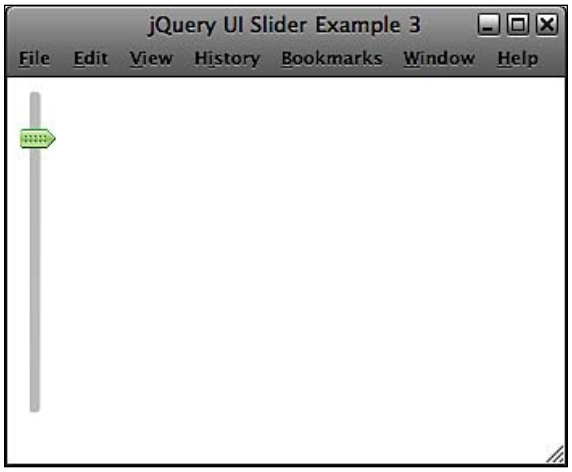
This new slider won't do anything interesting at this stage because it hasn't been configured. But you can see how easy it is to override the default styling to create your own unique slider implementation.

The slider widget has a handy feature built into it. The slider will automatically detect whether you wish to implement it horizontally or vertically. To make a vertical slider, all we need to do is use some custom images and change a couple of CSS rules. The widget will do the rest.

In `slider2.html`, remove the background `<div>` that we added for our custom background image and change the stylesheet link from `sliderTheme.css` to `verticalSlider.css`. Save these changes as `slider3.html`. The selectors and style rules in `verticalSlider.css` will need to be as follows:

```
.ui-slider, .ui-slider-1 {
  background:url(../img/slider/slider-bg-vert.png) no-repeat 6px 0px;
}
.ui-slider {
  height:200px; width:23px;
}
.ui-slider-handle {
  background:url(../img/slider/slider-handle-vert.gif) no-repeat;
  height:12px; width:23px;
}
```

When you launch the page, you'll see that the slider operates exactly as it did before, except that it now moves along the `y` axis . You can see this in the following screenshot:



# Configurable properties

Additional functionality, such as vertical sliders, multiple handles, and stepping, can also be configured using a literal object passed into the constructor method when the slider is initialized. The complete range of properties that can be used in conjunction with the slider widget are listed below:

| Property | Default Value | Usage |
|---|---|---|
| `animate` | `false` | Enables a smooth animation of the slider handle when the track is clicked |
| `axis` | | Sets the orientation of the slider if auto-detect fails |
| `handle` | `"ui-slider-handle"` | Sets the class name of the slider handle |
| `handles` | `{}` | Sets the boundaries for the slider handle(s) |
| `max` | `100` | Sets the maximum value of the slider |
| `min` | `0` | Sets the minimum value of the slider |
| `range` | `false` | Creates a styleable range between two slider handles |
| `startValue` | `0` | Sets the value the slider handle will start on |
| `stepping` | | Sets the distance between steps |
| `steps` | | Sets the number of steps |

The `stepping` and `steps` properties are very similar in usage but should not be confused. `stepping` is the distance between steps that the handle must move during each jump. `steps` refers to the number of steps, not the distance between them. These two properties should not be used together in the same implementation.

Let's put some of these properties to work. Apart from the default slider background used by the `flora` theme, a second background with defined step marks is also provided. We can use this in conjunction with the `stepping` and `steps` properties. In a new file in your text editor, create the following page:

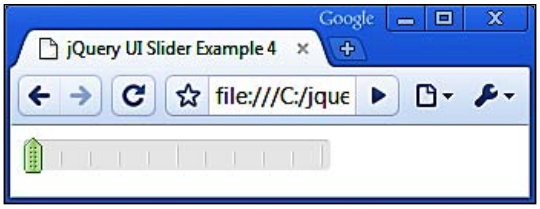
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <link rel="stylesheet" type="text/css" href="styles/
steppedSlider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
```

```
    <title>jQuery UI Slider Example 4</title>
  </head>
  <body>
    <div id="mySlider"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create config object
        var sliderOpts = {
         steps: 10
        };

        //create the slider
       $("#mySlider").slider(sliderOpts);

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

Save this as `slider4.html`. We're linking to a custom stylesheet for this example, which we'll need to create next. In another new page in your text editor, add the following rule:

```
.ui-slider, .ui-slider-1 {
  background-image:url(../jqueryui1.6rc2/themes/flora/i/
slider-bg-2.png);
}
```

Make sure this is saved in the `styles` folder as `steppedSlider.css`. When you run the example in a browser, you'll see that the slider background has ten visible step marks since we set the `steps` property to `10`. Now when you move the slider, it jumps from step mark to step mark, making the slider digital instead of analogue. The next screenshot shows how this example looks:



The `stepping` property achieves the same result as the `steps` property, but it does it in a different way. Let's see how this is done. Change the last `<script>` block in `slider4.html` so that it appears as follows:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){

    //create config object
    var sliderOpts = {
     stepping: 10
    };

    //create the slider
    $("#mySlider").slider(sliderOpts);
  });
</script>
```

Save this as `slider5.html`. We still provide `10` as the value even though we are now using the `stepping` property. We do this because the current maximum value for the slider is `100` (the default) and there are 10 step marks. So, 100 divided by 10 is 10. If we were to set the maximum value to `200` but still used the same image for the slider background, we would set `stepping` to `20` instead. The `steps` property would stay at `10` regardless of the maximum value.

The `startValue` property is just as easy to use. Depending on what we want the slider to represent, the starting value of the handle may not be `0`. If we wanted the handle to say, start at half way across the track instead of at the beginning, we could use the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <link rel="stylesheet" type="text/css" href="styles/
steppedSlider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 6</title>
  </head>
  <body>
    <div id="mySlider"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create config object
        var sliderOpts = {
          startValue: 50
        };

        //create the slider
        $("#mySlider").slider(sliderOpts);

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

Save this file as `slider6.html`. When the file is loaded in a browser, you'll see that the handle starts with a value of `50` instead of `0`.

# Using slider's callback functions

In addition to the properties we saw earlier, there are an additional four that can be used to define functions which are executed at different times during any slider interaction. This allows us to react to the events fired by the widget. These function properties are listed below:

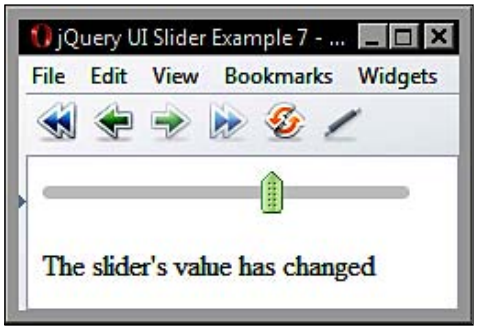
| Function | Usage |
|---|---|
| `change` | Called when the slider handle stops and its value has changed |
| `slide` | Called every time the slider handle moves |
| `start` | Called when the slider handle starts moving |
| `stop` | Called when the slider handle stops |

Hooking into these built-in callback functions is extremely easy. Let's put a basic example together that utilizes them all. In a new file in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 7</title>
  </head>
  <body>
    <div id="mySlider"></div><br>
    <div id="messageBox"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create config object
        var sliderOpts = {
          change: function() {
            var message = "The slider's value has changed";
            $("#messageBox").text(message);
          },
          slide: function() {
            var message = "The slider is sliding";
            $("#messageBox").text(message);
          },
          start: function() {
            var message = "The slider has started";
            $("#messageBox").text(message);
          },
          stop: function() {
            var message = "The slider has stopped";
         $("#messageBox").text(message);
          }
        };

        //create the slider
        $("#mySlider").slider(sliderOpts);
      });
    </script>
  </body>
</html>
```

Save this as `slider7.html`. It's a very basic page. We have a simple container `<div>` below the slider which will be used to hold a message. In the `<script>`, we define our callback functions within the configuration object. Then pass it into the constructor method as normal. Each time the state of the slider changes, the appropriate message will be written to the message box. The following screenshot shows a message:




You'll only see either the `slide` message or the `change` message because the `start` and `stop` messages get overwritten straight away, but they do occur. This example also shows us the order in which these functions will be executed:

- `start`
- `slide`
- `stop`
- `change`

Instead of simply writing a message to the page, these functions allow us to react appropriately to visitor interactions with the slider. We'll look at a more beneficial use of these properties later in the chapter.

# Slider methods

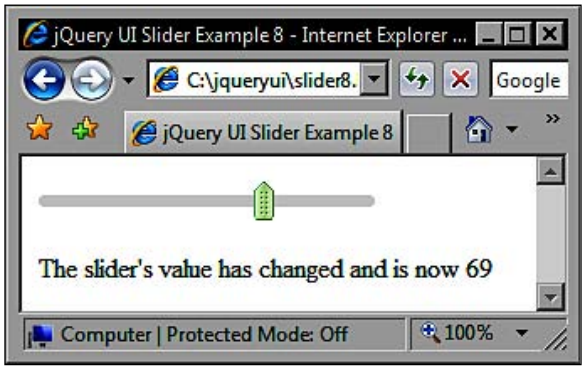
The slider is intuitive and easy to use, but to get any kind of workable result out of it, beyond what we've looked at so far, we'll need to make use of the methods that are built into it. The methods we can use are shown in the following table:

| Method | Used For/To |
|---|---|
| `moveTo` | Move the thumb to the specified value on the track |
| `value` | Retrieve the current value of the thumb |
| `disable` | Disable the functionality of the slider |
| `enable` | Enable the functionality of the slider |
| `destroy` | Return the underlying mark-up to its original state |

The first two methods, `moveTo` and `value`, are the most specific to the slider and are essential for working with it in any sensible way. To be able to use the slider widget effectively, you'll need to at least use the `value` method to obtain the position of the thumb following an interaction. Let's look at using this method next. Open `slider7.html` and change the final `<script>` block so that it appears as follows:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){

   //create config object
   var sliderOpts = {

     change: function() {

       //get the new value
       var val = $(this).slider("value");

       var message = "The slider's value has changed and is now " +
val;
       $("#messageBox").text(message);
     },
     slide: function() {
       var message = "The slider is sliding";
       $("#messageBox").text(message);
     },
     start: function() {
       var message = "The slider has started";
       $("#messageBox").text(message);
     },
     stop: function() {
       var message = "The slider has stopped";
       $("#messageBox").text(message);
     },
     steps: 100
    };

    //create the slider
    $("#mySlider").slider(sliderOpts);
  });
</script>
```

Save this file as `slider8.html`. This time when we move the slider, the new value is returned from the `value` method and is written to the message `<div>`. We set the `steps` property to `100`, or equal to the current maximum value, to avoid having a number with many decimal places returned by the method. Here's how it should look:

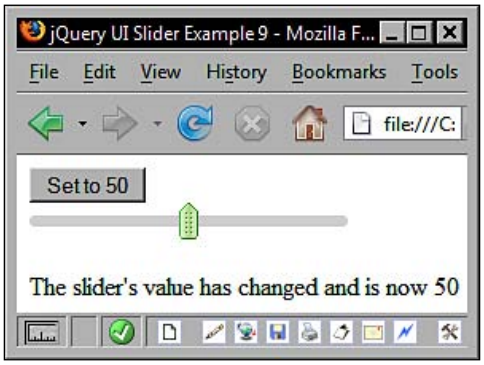



Using the `moveTo` method is just as quick and easy. If we add a `<button>` to the page, we can set it up so that clicking it moves the slider handle to a predefined point on the track. Change `slider8.html` so that it is the same as the following page:

```
!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 9</title>
  </head>
  <body>
    <button id="move">Set to 50</button><br>
    <div id="mySlider"></div><br>
    <div id="messageBox"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
```

```
        $(function(){

          //create config object
          var sliderOpts = {
           change: function() {

            //get the new value
            var val = $(this).slider("value");

             var message = "The slider's value has changed and is now " +
val;
           $("#messageBox").text(message);
           },
          steps: 100
         };

          //create the slider
         $("#mySlider").slider(sliderOpts);

        //define click handler for button
         $("#move").click(function(e, ui){

           //set slider value
           $("#mySlider").slider("moveTo", 50);

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

Save this as `slider9.html`. To use the `moveTo` method, all we do is pass in an additional argument specifying the new value. We've removed all of the event callbacks except for one, as using them all together in conjunction with this method can cause problems. Using the `change` event on its own is fine, however, as shown in the following screenshot:



# Slider animation

The slider widget comes with a built-in animation that moves the slider handle smoothly to a new position when the slider track is clicked. This animation is switched off by default. However, we can easily enable it by setting the `animate` property to `true`. Change the final `<script>` in `slider9.html` so that it is as follows:

```
<script type="text/javascript">
  //define function to be executed on document ready
  $(function(){

    //define config object
    var sliderOpts = {
      animate: true
    };

    //create the slider
    $("#mySlider").slider(sliderOpts);

  });
</script>
```

Save this version as `slider10.html`. The difference this property makes to the overall effect of the widget is extraordinary. Instead of the slider handle just moving instantly to a new position when the track is clicked, it smoothly slides there.

# Multiple handles

I mentioned earlier that a slider may have multiple handles. Implementing this feature couldn't be any easier thanks to another of the slider's built-in auto-detection features.

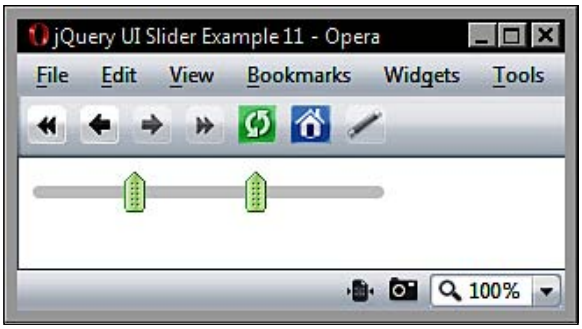
There are two ways you can implement multiple handles. First, you can supply the required number of child `<div>` elements within the element that is to be made a slider and give them all the class name `ui-slider-handle`. The second way is to give each of the child `<div>` elements their own collective class name and use the `handle` property to specify your class name as the class that should be applied to all handles.

The first method is probably the easiest as you don't need to worry about supplying new images for the handles. Let's take a look at a brief example. In a new page in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 11</title>
  </head>
  <body>
    <div id="mySlider">
      <div class="ui-slider-handle"></div>
      <div class="ui-slider-handle"></div>
    </div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create the slider
      $("#mySlider").slider();

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

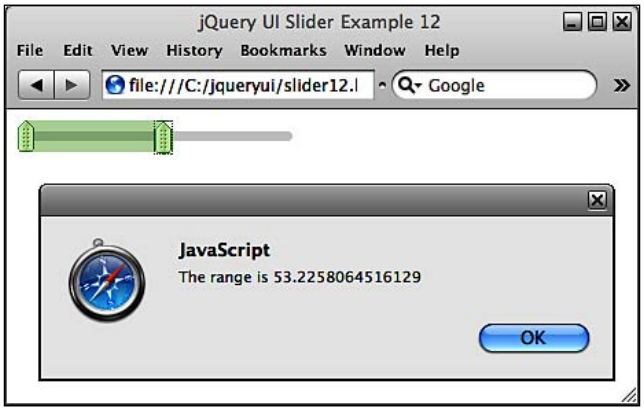
Save this as `slider11.html`. All we've done is insert two new `<div>` elements within our slider container element. The widget has created both new handles for us and, as you'll see, they both function exactly as a single handle does. The following screenshot shows our dual-handled slider:



When working with multiple handles, we can set the `range` property to `true`. This adds a styled range element between two handles. When the `range` property is `true`, we can also return the amount of range between the two handles using the second object (`ui`) which is automatically passed to our callback functions. In `slider10.html`, add the following additional code (shown in bold):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slider Example 12</title>
  </head>
  <body>
    <div id="mySlider">
      <div class="ui-slider-handle"></div>
      <div class="ui-slider-handle"></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //define config object
      var sliderOpts = {
        range: true,
        change: function(e, ui) {
          alert("The range is " + ui.range);
       }
      };

       //create the slider
      $("#mySlider").slider(sliderOpts);

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

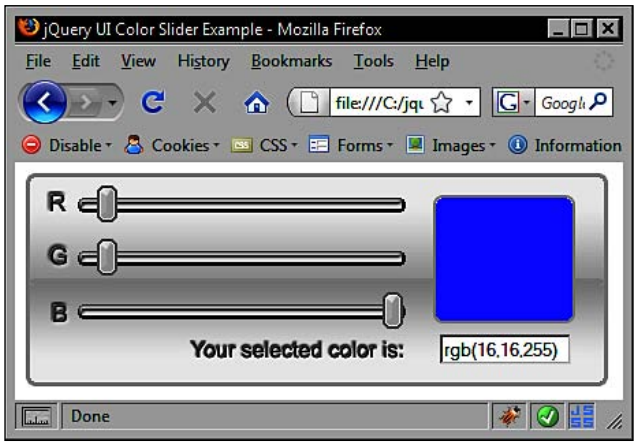
Save this as `slider12.html`. Now when you move one of the handles, you should see a semi-opaque overlay between the two handles. Also, when you drop the handle, you'll get a lovely intrusive alert telling you the range. The alert is just for the purpose of this example and should not be used in the wild! The next screenshot shows both the range element and the alert:




Unfortunately, setting the `steps` property to `100` to return an integer for our alert does not work when the `range` property is enabled.

# Fun with slider

A fun implementation of the slider widget, which could be very useful in certain applications, is the color slider. Let's put what we've learned of this widget so far together to produce a versatile and easily created color choosing tool. The following screenshot shows what we'll be making:



In a new file in your text editor, begin with the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.slider.css">
    <link rel="stylesheet" type="text/css" href="styles/
ColorSliderTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Color Slider Example</title>
  </head>
  <body>
    <div id="container">

      <div id="rSlider"></div><br>
      <div id="gSlider"></div><br>
      <div id="bSlider"></div>
      <input id="output" type="text">
    </div>
    <div id="colorBox"></div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.slider.js"></script>
<script type="text/javascript">
    //define function to be executed on document ready
      $(function(){

        //initialize values
        var r = 0;
        var g = 0;
        var b = 0;

        //create config obj
       var sliderOpts = {
          max: 255,
          steps: 255,
          slide: function(e, ui) {
            var val = $(this).slider("value");
            var id = $(this).attr("id");

            if (id == "rSlider") {
```

```
                r = val;
              } else if (id == "gSlider") {
                g = val;
              } else {
                b = val;
              }

              var rgbString = "rgb(" +r+ "," +g+ "," +b+ ")";

              $("#colorBox").css({
                  backgroundColor:rgbString
                });

              $("#output").val(rgbString);
            }

          };

          //create the dialog
          $("#rSlider, #gSlider, #bSlider").slider(sliderOpts);

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

Save this as `colorSlider.html`. The page itself is simple enough. We've got some elements used primarily for displaying the different components of the color slider, as well as the individual container elements which will be transformed into slider widgets.

The JavaScript is just as simple. We set the initial RGB values of each slider to `0` using the first three variables. `0,0,0` is black in RGB of course. However, if we don't set these variables initially, the `colorBox` won't change color until all three sliders have been moved.

As RGB color values range from 0 to 255, we set the `max` property to `255` in our configuration object. We also set the `steps` property to 255 as well to make sure we get whole numbers only.

The `change` callback is where it all happens. Every time a handle is dropped we update the appropriate variable and then construct an RGB string from the values of our variables. This is necessary as we can't pass the variables directly into jQuery's `css` method.

We'll need some CSS as well to complete the overall appearance of the control. In a new page in your text editor, create the following stylesheet:

```
#container {
  width:426px; height:156px;
  background:url(../img/color-slider/colorSlider_bg.gif) no-repeat;
  position:relative;
  z-index:1;
}
.ui-slider {
  width:240px; height:11px;
  background:url(../img/color-slider/colorSlider_track.gif) no-repeat;
  position:relative; top:18px; left:38px;
  margin-bottom:8px;
}
.ui-slider-handle {
  width:15px; height:27px;
  background:url(../img/color-slider/colorSlider_handle.png)
  no-repeat;
  margin-top:-9px;
}
#colorBox {
  width:104px; height:94px;
  background:#ffffff url(../img/color-slider/color_box.gif) no-repeat;
  position:absolute; left:306px; top:24px;
  z-index:2;
}
#output {
  position:absolute; bottom:16px; right:27px; width:92px;
}
```

Save this as `colorSliderTheme.css` in the `styles` folder. When we run the example, we should find that everything works as expected. As soon as we start moving any of the sliders, the color box color channel we are changing is reflected by the color of the box. We can also write the value to a text box to see the actual RGB value of the color we have selected.

# Summary

In this chapter, we looked at the slider widget and saw how quickly and easily it can be put on the page. It requires minimal underlying mark-up and just a single line of code to initialize. We also saw how it intelligently detects whether to implement as a horizontal or vertical slider.

We looked at properties that can be used to control how the slider behaves and how it can be fine-tuned to suit a range of implementations. There are properties to configure whether the slider should move smoothly across the track, or jump along a series of predefined steps, and to configure a non-zero starting value for the handle.

Configured as properties of the widget, we also saw the rich event model that can easily be hooked into, and reacted to, with up to four separate callback functions . This allows us to execute code at important times during an interaction.

Finally, we looked at the range of methods that can be used to programmatically interact with the slider. Moving the handle to a specified point on the track, for example.

These properties and methods turn the widget into a useful and highly functional interface tool that adds an excellent level of interactivity to any page.

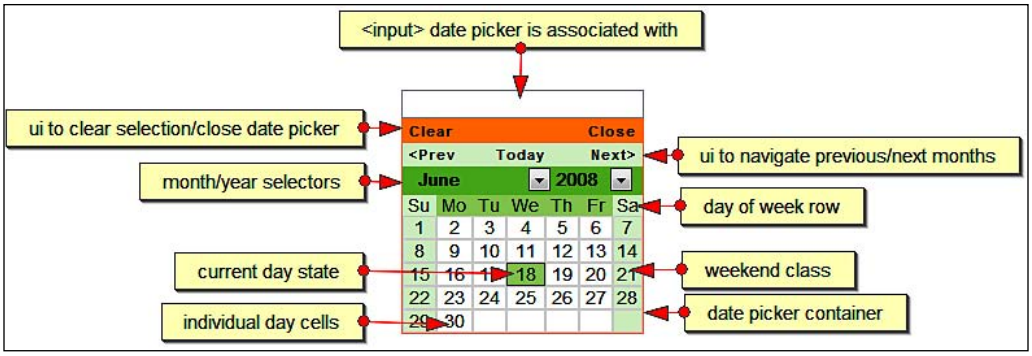
# 6
# Date Picker

The UI date picker widget is probably the most refined and documented widget found in the jQuery UI library. It should be no surprise to learn that it's highly configurable and extremely easy to implement and customize. It was written by Marc Grabanski and Keith Wood in 2005. The current version at the time of writing is 3.4.3.

Quite simply, the date picker widget provides an interface which allows visitors to your site or application to select dates. Wherever a form field is required which asks for a date to be entered, the date picker widget can be added. This means your visitors get to use an attractive, engaging widget and you get dates in the format in which you expect them.

Additional functionality built into the date picker includes automatic opening and closing animations and the ability to navigate the interface of the widget using the keyboard. While holding the down *Ctrl* key (or Apple key on the Mac), the arrows on the keyboard can be used to choose a new day cell, which can then be selected using the *Return* key.

While easy to create and configure, the date picker is a complex widget made up of a wide range of underlying elements, as the following screenshot shows:



Despite this complexity, we can implement the default date picker with just a single line of code, much like the other widgets in the library that we have covered so far. During this section, we will look at the following subjects:

- A default date picker implementation
- Changing the date picker's appearance
- Exploring the configurable properties
- The new and improved `dateFormat` property
- Easy internationalization
- Implementing a trigger button
- Multiple month date pickers
- Enabling range selection
- Configuring alternative animations
- Making things happen with date picker's methods
- Using AJAX with the date picker

# The default date picker

To create a default date picker, the following code meets the minimum requirements:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 1</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
```

```
        //create the date picker
        $("#date").datepicker();
    });
  </script>
 </body>
</html>
```

Save this as `datepicker1.html`. On the page, all we have is a `<label>` and a standard text `<input>` element. We don't need to specify any empty container elements for the date picker widget to be rendered into. All of the required mark-up to produce the widget is added automatically by the library.

The JavaScript is equally as simple. We use the `$(function() { });` construct to execute some code when the page loads. The code that we execute is the `datepicker` constructor method, which is called on a jQuery object representing our `<input>` field.

When you run the page in your browser and focus the `<input>` element, the default date picker should appear below the input and should look like the screenshot at the start of the chapter.

Apart from great looks, the default date picker also comes with a lot of built-in functionality. When the calendar opens, it is smoothly animated from zero to full size, it will automatically be set to the present date, and selecting a date will automatically add the date to the `<input>` and close the calendar (again with a nice animation).

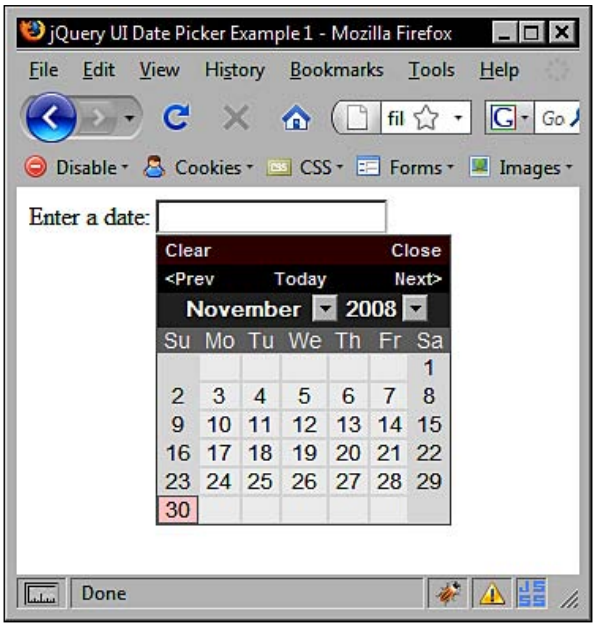
With no additional configuration, and a single line of code, we now have a perfectly usable and attractive widget making date selection easy. If all you want to do is let people pick a date, this is all you need to do.

The source files required for the default date picker implementation are:

- The latest version of jQuery (1.2.6 at the time of writing)
- The UI library core file `ui.core.js`
- The date picker source file `ui.datepicker.js`

# Skinning the date picker

The date picker comes with two themes. The first is the `flora` theme, which is shown in the screenshot at the start of this chapter, and the `default` theme, which is shown in the following screenshot:




Both of these themes not only control how the widget looks, but also ensure that it displays correctly. We can easily override specific selectors with our own stylesheet to easily change how the widget looks but not how it works. Let's do this next. In a new file in your text editor, add the following stylesheet:

```
#ui-datepicker-div { border:1px solid #3399ff; }
#ui-datepicker-div a, .ui-datepicker-inline a {
  color:#ffffff !important;
}
#ui-datepicker-div a:hover, .ui-datepicker-inline a:hover {
  color:#000000 !important;
  background-color:#ffffff !important;
}
.ui-datepicker-header { background:#000000; }
.ui-datepicker-header select {
  background-color:#3399ff; font-size:70%; width:72px;
  color:#ffffff;
}
```

```
.ui-datepicker-control { background-color:#3399ff; }
.ui-datepicker-control, .ui-datepicker-links {
  font-size:70%;
}
.ui-datepicker-control a, .ui-datepicker-links a {
  color:#ffffff !important;
}
.ui-datepicker-control a:hover { color:#000000 !important; }
.ui-datepicker-links {
  background-color:#000000; color:#ffffff;!important;
}
.ui-datepicker .ui-datepicker-title-row {
  background-color:#000000; color:#ffffff; font-size:90%;
}
.ui-datepicker .ui-datepicker-title-row td {
  border-bottom:1px solid #3399ff;
}
.ui-datepicker .ui-datepicker-week-end-cell {
  background-color:#000000;
}
.ui-datepicker .ui-datepicker-days-row {
  background-color:#000000;
}
.ui-datepicker .ui-datepicker-days-cell {
  border:0; border-right:1px solid #3399ff;
  border-bottom:1px solid #3399ff; color:#ffffff !important;
}
.ui-datepicker-today, .ui-datepicker .ui-datepicker-days-cell-over {
  background-color:#99ccff !important;
  color:#ff0000 !important;
}
.ui-datepicker-one-month { width:145px !important; }
table.ui-datepicker { font-size:80%; }
```

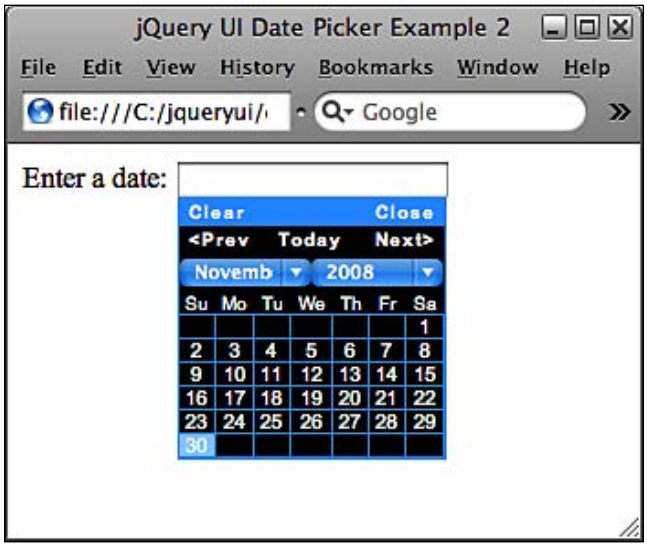
Save this file as `datePickerTheme.css` in your `styles` folder. Create a new folder inside the `img` folder and name it `date-picker`. Also, unpack the images from the code download for the chapter into the new folder. These selectors use the same specificity as the original selectors in the theme file to target particular elements with pin-point precision. Because our stylesheet will be added after the skin file, our new rules will take precedence. We need the `!important` rule occasionally to override styles added as element attributes by the widget.

This is probably the minimum amount of code that is needed to change the aesthetical appearance of the widget, and it's quite a basic change in style. There is much more that could be done to really make the different elements stand out.

Don't forget to link to the new stylesheet in the `<head>` of the page, directly after the skin file, with the following code:

```
<link rel="stylesheet" type="text/css" href="styles/
datePickerTheme.css">
```

Save this as `datepicker2.html`. Once this is done, you should have something similar to that shown in the following screenshot:




Don't forget that in addition to creating our own stylesheet which overrides specific style rules, we can also use Theme Roller to produce a comprehensive custom theme for the widget in no time at all.

# Configurable properties of the picker

The date picker has a large range of configurable properties, currently sixty six to be exact, which is more than twice the number that any of the other UI library components have. The following table lists the basic properties, their default values, and gives a brief description of their usage:

| Property | Default Value | Usage |
|---|---|---|
| altField | "" | Specify a selector for an alternative <input> field which the selected date is also added to |
| altFormat | "" | Specify an alternative format for the date added to the alternative <input> |
| appendText | "" | Add text after the date picker <input> to show the format the selected date will have |
| buttonImage | "" | Specify a path to the image to use for the trigger <button> |
| buttonImageOnly | false | Set to true to use an image instead of a trigger button |
| buttonText | "..." | Text to display on a trigger <button> (if present) |
| changeFirstDay | true | Reorder the calendar when a day heading is clicked |
| changeMonth | true | Show the month change drop-down |
| changeYear | true | Show the year change drop-down |
| closeAtTop | true | Show the **close** button at the top of the calendar |
| constrainInput | true | Constrain the <input> to the format of the date |
| defaultDate | null | Set the date that is initially highlighted |
| duration | normal | Set the speed at which the date picker opens |
| goToCurrent | false | Set the current day link to move the date picker to currently selected date instead of today |
| hideIfNoPrevNext | false | Hide the **Prev**/**Next** links when not needed |
| highlightWeek | false | Set the date picker to highlight the entire row when a day is hovered over |
| isRTL | false | Set the calendar to right-to-left format |
| mandatory | false | Enforce date selection |
| maxDate | null | Set the maximum date that can be selected |
| minDate | null | Set the minimum date that can be selected |
| navigationAsDateFormat | false | Allows us to specify month names as the **Prev**, **Next**, and **Current** links |
| numberOfMonths | 1 | Set the number of months shown on a single date picker |

| Property | Default Value | Usage |
|---|---|---|
| `rangeSelect` | `false` | Enable the selection of date ranges |
| `rangeSeparator` | `"-"` | Set the separator between dates when using ranges |
| `shortYearCutoff` | `"+10"` | This is used to determine the current century when using the `y` year representation and numbers less are deemed to be in the current century |
| `showOn` | `"focus"` | Set the event on which to show the date picker |
| `showOtherMonths` | `false` | Show the last and first days of the previous and next months |
| `showStatus` | `false` | Show a status bar within the date picker |
| `showWeeks` | `false` | Show the week number column |
| `showAnim` | `show` | Set the animation that is performed when the date picker is opened or closed |
| `showOptions` | `{}` | Set additional animation configuration options |
| `stepMonths` | `1` | Set how many months are navigated with the **prev** and **next** links |
| `yearRange` | `"-10+10"` | Specify the range of years in the year drop-down |

A number of examples will be needed to look at just some of the available properties. Once we've got the hang of using a few of them, the rest will become just as easy to make use of. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 3</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //define config object
        var pickerOpts = {
          appendText: " MM/DD/YYYY",
          changeFirstDay: false,
          changeMonth: false,
          changeYear: false,
          closeAtTop: false,
          showOtherMonths: true,
          showStatus: true,
          showWeeks: true,
          duration: "fast"
        };
        //create the date picker
        $("#date").datepicker(pickerOpts);
      });
    </script>
  </body>
</html>
```

Save this as `datePicker3.html`. We've reverted back to the `flora` theme so we can clearly see how the changes we've made affect the widget. The following image shows how the widget will look after configuring these properties:



We've used a number of properties in this example. The appearance of the initial page has been changed using the `appendText` property, which has added the specified text string directly after the `<input>` field the picker is associated with. This helps visitors to clarify the format of the date that is expected.

For styling purposes, we can target this new string using the `ui-datepicker-append` selector in a stylesheet if necessary, as this is the class name given automatically to the specified text.

When one of the day headings is clicked, the date picker columns will no longer rearrange themselves, thanks to setting the `changeFirstDay` property to `false`. The month and year drop-downs have been removed by setting the `changeMonth` and `changeYear` properties to `false`.

Setting the `closeAtTop` property to `false` has of course moved the **Close** button to the bottom of the date picker, but it's also taken the **Clear** button with it. The functionality of these remain the same, all that has changed is their position.

By setting the `showOtherMonths` property to `true`, we've added greyed-out (non-selectable) dates from the next month to the empty squares that sit at the bottom of the date picker after the current month ends.

The addition of the status bar, configured using `showStatus`, above the bottom row, is a great way of providing usage information to visitors in a non-intrusive manner. But remember that this can bring much more additional configuration when working with non-standard locales.

We've also added a new column to the body of the date picker using the `showWeeks` property. Now the first row of the date picker shows the corresponding week number in each week's row.

The speed at which the widget opens and closes is visibly quicker thanks to the use of the `duration` property. This property requires a simple string and can take values of either `slow`, `normal`, or `fast`, with the default being `normal`.

# Changing the date format

The `dateFormat` property is one of the localization properties at our disposal for advanced date picker locale configuration. Setting this property allows you to quickly and easily set the format of dates using a variety of short-hand references. The format of dates can be any combination of any of the following references (they are case-sensitive):

- `d` – day of month (single digit where applicable)
- `dd` – day of month (two digits)

- `m` – month of year (single digit where applicable)
- `mm` – month of year (two digits)
- `y` – year (two digits)
- `yy` – year (four digits)
- `D` – short day name
- `DD` – full day name
- `M` – short month name
- `MM` – long month name
- `''` – any literal text string

We can use these to quickly configure our preferred date format, as in the following example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 4</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jquery.ui-1.5b4/
jquery-1.2.4b.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //define config object
        var pickerOpts = {
          dateFormat: "d MM yy"
        };
        //create the date picker
```

```
            $("#date").datepicker(pickerOpts);
        });
    </script>
  </body>
</html>
```

Save this as `datepicker4.html`. We use the `dateFormat` property to specify a string containing the short-hand date codes for the selected date. The format we set shows the day of the month (using a single digit if possible) with `d`, the full name of the month with `MM`, and the full four-digit year with `yy`.

When dates are selected and added to the associated `<input>`, they will be in the format specified in the configuration object, as in the following screenshot:




When using the literal text option to configure dates, any letters used as short-hand for the different formats will need to be 'escaped' using single quotes. For example, to add the string `Selected:` to the start of the date, you would need to use the string `Selecte'd':` to avoid having the lowercase `d` picked up as the short day of month format.

# Localization

In addition to the properties already listed, there is also a range of regionalization properties used to provide locale support in order to easily display date pickers with all of the text shown in an alternative, non-standard language.

Those properties that are used specifically for the localization of textual elements of the widget are listed below:

| Property | Usage |
|---|---|
| `clearText` | Text to display on the **Clear** button |
| `clearStatus` | Text to display in the status bar for the **Clear** link on hover |
| `closeText` | Text to display on the **Close** link |
| `closeStatus` | Text to display in the status bar for the **Close** link on hover |

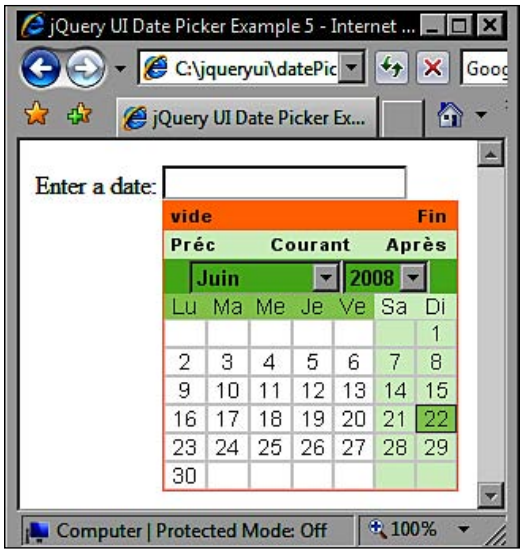
| Property | Usage |
|---|---|
| currentText | Text to display on the **Current** link |
| currentStatus | Text to display in the status bar for the **Current** link on hover |
| dateFormat | The format selected dates should appear in <input> |
| dateStatus | Text to display in the status bar for date links on hover |
| dayNames | An array of day names |
| dayNamesShort | An array of abbreviated day names |
| dayNamesMin | An array of 2-letter day names |
| dayStatus | Text to display in the status bar for the day of the week link on hover |
| firstDay | Specify the first column of days in the date picker |
| initStatus | Text to display in the status bar when date picker opens |
| monthNames | An array of month names |
| monthNamesShort | An array of abbreviated month names |
| monthStatus | Text to display in the status bar for the month drop down on hover |
| nextStatus | Text to display in the status bar for the **Next** link on hover |
| nextText | Text to display on the **Next** link |
| prevStatus | Text to display in the status bar for the **Prev** link on hover |
| prevText | Text to display on the **Prev** link |
| weekHeader | Text to display in the column header for the week of the year |
| weekStatus | Text to display in the status bar for the week of the year column header on hover |
| yearStatus | Text to display in the status bar for the year drop-down on hover |

Localization is very easy to configure using a standard configuration object containing the required properties from the table above and the values that you would like to use. In this way, any alternative language can be implemented. For example, to implement a French date picker, we could use the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 5</title>
  </head>
  <body>
    <div class="row"><label>Enter a date: </label><input id="date">
</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //define config object for localisation
      var frenchOpts = {
        clearText: "vide",
        closeText: "Fin",
        currentText: "Courant",
        dayNamesMin: [ "Lu", "Ma", "Me", "Je", "Ve", "Sa", "Di" ],
        firstDay:1,
        monthNames: [ "Janvier", "Février", "Marche", "Avril",
"Pouvez", "Juin", "Juillet", "Août", "Septembre", "Octobre",
"Novembre", "Décembre" ],
        nextText: "Après",
        prevText: "Préc",
     };
      //create the date picker
      $("#date").datepicker(frenchOpts);
     });
   </script>
  </body>
</html>
```

Save this file as `datePicker5.html`. Most of the properties are used to provide simple string substitutions. However, the `dayNamesMin` and `monthNames` properties require arrays. Notice that the `dayNamesMin`, and other day-name related arrays, should begin with Sunday (or the localized equivalent). Now when you run the page in your browser, all of the default text on the date picker should be in French, as in the following screenshot:






The translations used in this example were provided by the Yahoo! Babel Fish service at `http://uk.babelfish.yahoo.com/`.

Only eight properties are required to internationalize the default date picker. However, when using additional non-default UI elements, like the status bar, an additional fourteen properties are required, making the required object literal look like this:

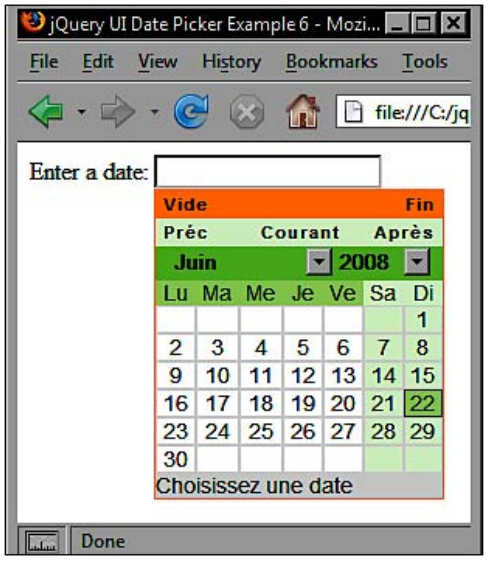
```
//define config object for localisation
var frenchOpts = {
  clearText: "Vide",
  clearStatus: "Effacez la date du jour",
  closeText: "Fin",
  closeStatus: "Fermez-vous sans changement",
  currentText: "Courant",
  currentStatus: "Montrez le mois courant",
  dateStatus: "Choisi DD, d M",
  dayStatus: "Commencez semaine avec DD",
  dayNames: [ "Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi",
"Vendredi", "Samedi" ],
  dayNamesMin: [ "Di", "Lu", "Ma", "Me", "Je", "Ve", "Sa" ],
  firstDay: 1,
  initStatus: "Choisissez une date",
  monthNames: [ "Janvier", "Février", "Marche", "Avril", "Pouvez",
"Juin", "Juillet", "Août", "Septembre", "Octobre", "Novembre",
"Décembre" ],
```

```
    monthNamesShort: [ "Jan", "Fév", "Mar", "Avr", "Pou", "Jui", "Jui",
"Aoû", "Sep", "Oct", "Nov", "Déc" ],
    monthStatus: "Montrez un mois différent",
    nextText: "Après",
    nextStatus: "Montrez le mois prochain",
    prevText: "Préc",
    prevStatus: "Montrez le mois précédent",
    weekStatus: "",
    showStatus: true,
    yearStatus: "Montrez une année différente"
};
```

This code change can be saved as `datePicker6.html`. Apart from the visible elements in the initial date picker, the addition of the status bar and the additional object properties (mostly those ending in `Status`) gives us the internationalized status bar messages when hovering over different date picker elements.

We've also used the short-hand `dateFormat` properties in this example. For the value of the `dateStatus` property in the above code, we supplied some normal text data within the string, as well as the short-hand properties `DD`, `d M`.

The following screenshot shows one of our internationalized messages:



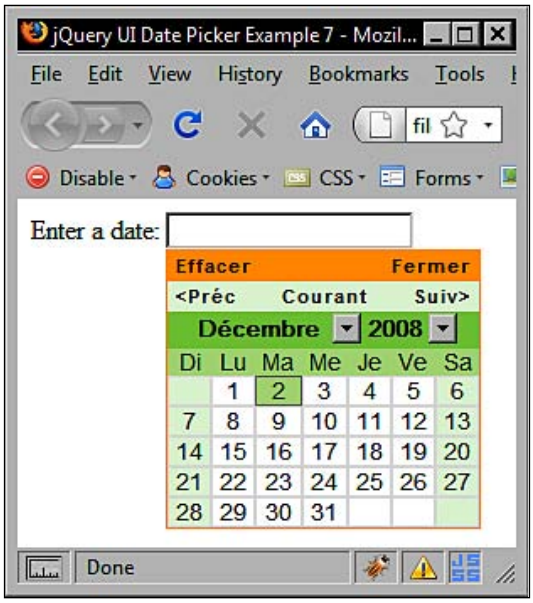

To make implementing internationalization easier, the creators of the date picker widget also provide a wide range of preconfigured locale packages, which are all included with the current release of the library. These language packs can be found in the `i18n` folder, within the `ui` folder in the library's directory hierarchy.

Let's see how easy it is to implement the date picker using an alternative language pack. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 7</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/i18n/
ui.datepicker-fr.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create the date picker
        $("#date").datepicker();
      });
    </script>
  </body>
</html>
```

Save this as `datePicker7.html`. We have still implemented a French date picker in this example, although we did so with considerably less code and much more accurate translations! All we do is include the `<script>` element linking to the language pack that we want to use, and then create the date picker using the standard constructor method.

That's it. The widget will automatically use the specified language with no additional configuration. Now, our French date picker will appear as follows:




# Callback properties

The final set of configurable properties relates to the event model exposed by the widget. It consists of a series of callback functions we can use to specify code to be executed at different points during an interaction with the date picker. These are listed below:

| Property | Usage |
|---|---|
| `beforeShow` | Accepts a configuration object that can be used to customize the date picker |
| `beforeShowDay` | Used to preselect specific days |
| `calculateWeek` | Change the calculation that is used to calculate the week of the year |
| `onSelect` | Set a callback function for the `select` event |
| `onChangeMonthYear` | Set a callback function to be executed when the current month or year changes |
| `onClose` | Set a callback function for the `close` event |
| `statusForDate` | The function to call to determine the status bar text for the date |

To highlight how useful these callback properties are, we can extend the previous internationalization examples to allow visitors to choose any available language of the date picker. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jquery.ui-1.5b4/
themes/flora/flora.datepicker.css">
    <link rel="stylesheet" type="text/css" href="styles/
intlPicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 8</title>
  </head>
  <body>
    <a id="uk" href="#" title="English"></a><a id="france" href="#"
title="Français"></a>
    <div class="row"><label>Enter a date: </label><input id="date">
</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/i18n/
ui.datepicker-fr.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //flag variable
       var locale = "en";
      $("a").click(function() {
        (this.id != "france") ? locale = "en" : locale = "fr";
      });
       function setLocale() {
         return (locale == "fr") ? $.datepicker.regional['fr'] :
$.datepicker.regional[''];
       }
       //create the date picker
       $("#date").datepicker({ beforeShow:setLocale });
     });
   </script>
  </body>
</html>
```

This can be saved as `datePicker8.html`. We have wrapped the `<label>` and `<input>` elements from the previous examples for styling purposes. We also added two new link elements which will be used as buttons to select the language of the date picker. We've also linked to the French language pack as we did before.

The first thing we do in the code in this example is set the `locale` variable that will be used to tell the date picker which language it should display. We set it to `en` as the default value.

Next, we set a click handler which will react to either of the buttons being clicked. The anonymous function looks at the `id` attribute of the anchor that was clicked and updates the value of the `locale` variable accordingly.

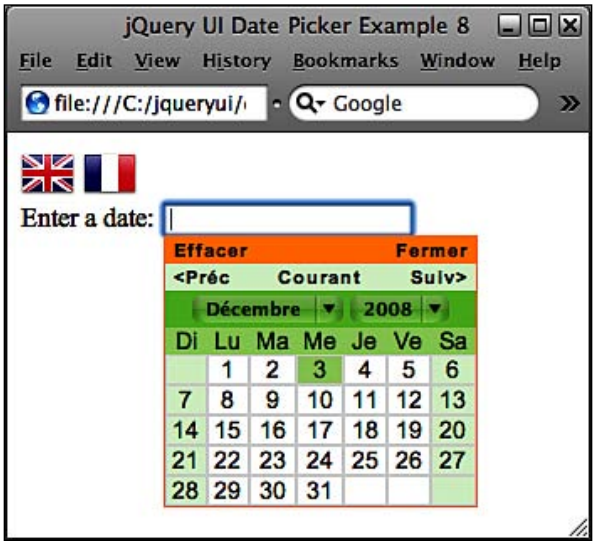
Following the click handler, we set a callback function called `setLocale`, which is used to return new settings for the date picker. When the date picker is initialized, the `$.datepicker` manager object is created with various properties to control how the date picker functions. We can use this manager object to update the locale of the date picker, which is exactly what we do in this example.

We've configured the `beforeShow` property of the date picker within the constructor function and specified our callback function as its value. The `beforeShow` event is fired just before the date picker is shown. When this happens, our callback function checks the value of the `locale` variable and shows the appropriate language by setting the `regional` property of the manager object. Note that only languages that have their language pack included in the page will work.

We also use a custom stylesheet for this example. It's relatively simple, consisting of the following selectors and rules:

```
#uk {
  background:url(../img/Uk.png) no-repeat;
  width:32px; height:32px; display:block; float:left;
  margin-right:5px;
}
#france {
  background:url(../img/France.png) no-repeat;
  width:32px; height:32px; display:block; float:left;
}
.row { clear:both; }
```

Save this in the `styles` folder as `intlPicker.css`. When you run this page in your browser, you should be able to set the locale, to either French or English, by clicking the corresponding flag. Here's how the page should look:



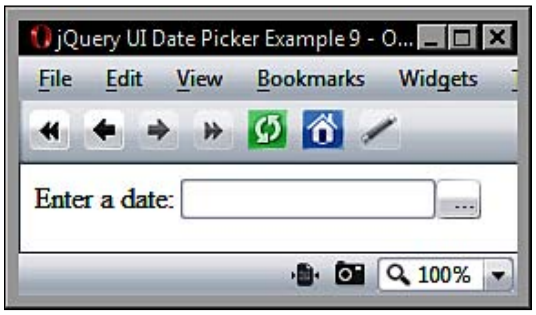

# Trigger buttons

By default, the date picker is opened when the `<input>` element it is associated with receives focus. However, we can change this very easily so the date picker opens when a `<button>` is clicked instead. The most basic type of `<button>` can be enabled with just the `showOn` property, which we can use with the next example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 9</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){

       //create config object
      var pickerOpts = {
        showOn: "button"
      };

      //create the date picker
      $("#date").datepicker(pickerOpts);
     });
   </script>
  </body>
</html>
```

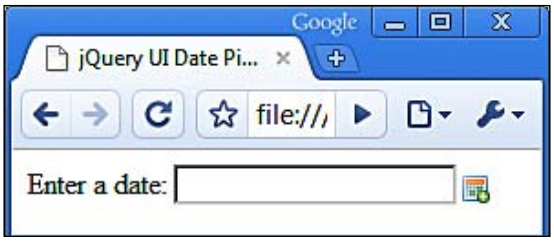
Save this as `datePicker9.html`. Setting the `showOn` property to `true` in our configuration object will automatically add a simple `<button>` element directly after the associated `<input>` element. The date picker will now only open when the `<button>` is clicked, rather than when the `<input>` is focused. The new `<button>` is shown in the following screenshot:




The text shown on the `<button>` (**...** in this example) can easily be changed by providing a new string as the value of the `buttonText` property. We can just as easily add an image to the `<button>` as well. Using either the `buttonImage` or `buttonImageOnly` properties, an image will be used instead of a traditional `<button>`. Change the configuration object so that it appears as follows:

```
var pickerOpts = {
  showOn: "button",
  buttonImage: 'img/date-picker/cal.png',
  buttonImageOnly: true
};
```

Save this as `datePicker10.html`. This should give you a nice image-only button. The `buttonImage` property allows us to specify the relative path of the image to use, and the `buttonImageOnly` property ensures that only the image is shown, not the image on top of the `<button>`. This is illustrated in the following screenshot:




You should note that when an image is used instead of a `<button>`, the value of the `buttonText` property is used as the `title` and `alt` attributes of the image.



The calendar icon used in this example was taken, with thanks, from the Silk Icon Pack by Mark James and is available at `http://www.famfamfam.com`.

# Multiple months

The date picker need not only display a single month. Instead, it can be configured to show multiple months. It takes just two properties to implement multiple months, although I prefer to use four properties. Create the following new page:

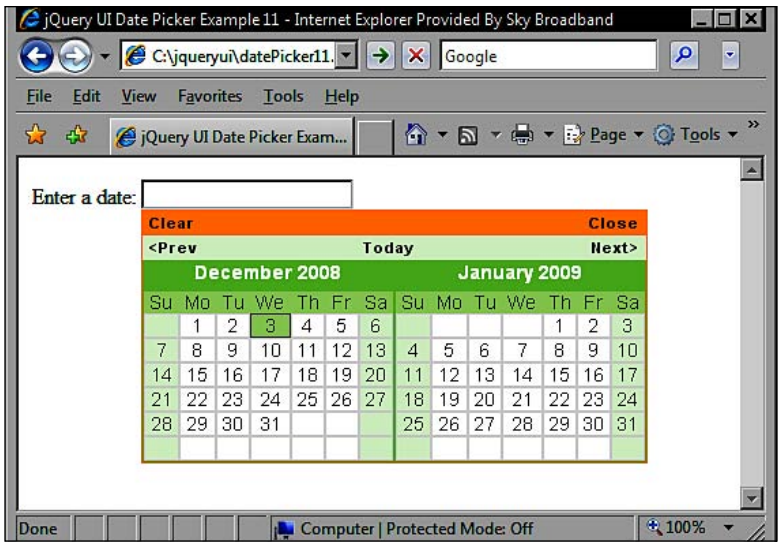
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 11</title>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
```

```
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //create config object
      var pickerOpts = {
        numberOfMonths: 2,
        stepMonths: 2,
        changeMonth: false,
        changeYear: false
      };
      //create the date picker
      $("#date").datepicker(pickerOpts);
     });
    </script>
  </body>
</html>
```

Save this as `datePicker11.html`. The `numberOfMonths` property takes an integer representing, of all things, the number of months you would like displayed at once. The `stepMonths` property controls how many months are changed when the **Prev** or **Next** links are used. This should be set to the same value as the `numberOfMonths` property.

By default, the `changeMonth` and `changeYear` drop-downs will be shown above the first month. Personally, I think the date picker looks better without these when using multiple months. So, we've removed them by setting the `changeMonth` and `changeYear` properties to `false`.

The date picker in this configuration will appear like this:



# Enabling range selection

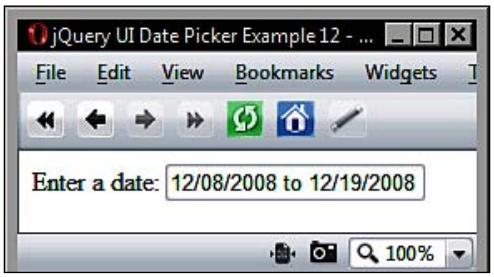
In some situations, you may want your visitors to be able to select a range of dates instead of a single date. Like everything else, the date picker widget makes configuring selectable ranges easy as we only need to work with two configuration properties. In a new file, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 12</title>
    <style type="text/css">
      input { width:180px; }
    </style>
  </head>
  <body>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
     //define function to be executed on document ready
     $(function(){
       //create config object
       var pickerOpts = {
         rangeSelect: true,
         rangeSeparator: " to "
       };
        //create the date picker
        $("#date").datepicker(pickerOpts);
       });
    </script>
  </body>
</html>
```

Save this as `datePicker12.html`. Enabling selectable ranges of dates, instead of single days, simply requires setting the `rangeSelect` property to `true`. Optionally, we can also change the text that is used as the separator between the two dates once they have been added to the `<input>` field. The default is a – character, but we have substituted this for the string `to` (with a space on either side).

We've also had to increase the width of the `<input>` element slightly so that it is wide enough to show all of the selected range. This has been done using a simple style rule added to the `<head>` of the page, which is purely for the purposes of this example. All style rules should normally go into their own stylesheet.

Once a date has been selected, the `<input>` element should appear as in the following screenshot:




The behavior of the widget changes slightly when range selection is enabled. Now, after a date is selected, the widget doesn't close instantly, it stays open and all dates prior to the selected date become unselectable.

The date picker closes once a second date has been selected and both the starting and ending dates are added to the `<input>` field along with the separator. If the date picker is opened a second time, the range that was selected is highlighted.

# Configuring alternative animations

The date picker API exposes two properties related to animations. These are the `showAnim` and `showOptions` properties. By default, the date picker uses the show effect to display the widget. This shows the date picker smoothly increasing in size and opacity.

However, we can change this, so that it uses any of the other effects included with the library (see chapter 12), which we'll do in the next example. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
```

```
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 13</title>
  </head>
  <body>
    <div class="row"><label>Enter a date: </label><input id="date">
</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.drop.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //define config object
        var pickerOpts = {
          showAnim: "drop",
          showOptions: { direction: "up" }
        };

        //create the date picker
        $("#date").datepicker(pickerOpts);
      });
    </script>
  </body>
</html>
```

Save this as `datepicker13.html`. To use alternative effects, we need to use two new `<script>` resources. These are the `effects.core.js` and the source file of the effect we wish to use in this example, `effects.drop.js`. We'll look at both of these effects in more detail in the last chapter, but they are essential for this example to work.

Our simple configuration object configures the animation to `drop` using the `showAnim` property, and sets the `direction` property of the effect using `showOptions`. When you run this example now, the date picker should slide down into position instead of opening. Other effects can be implemented in the same way.

# Date picking methods

Apart from the enormous number of properties at our disposal, there are also a number of useful methods defined that make programmatically working with the date picker a breeze. The date picker class defines the following methods:

| Method | Usage |
|---|---|
| `change` | Use a configuration object to change a pre-existing (attached) date picker |
| `destroy` | Disconnect and remove an attached date picker |
| `dialog` | Open the date picker in a dialog widget |
| `disable` | Disable an `<input>` field (and therefore the attached date picker) |
| `enable` | Enable a disabled `<input>` field (and date picker) |
| `getDate` | Get the currently selected date |
| `hide` | Programmatically close a date picker |
| `isDisabled` | Determine whether a date picker is disabled |
| `setDate` | Programmatically select a date |
| `show` | Programmatically show a date picker |

The `change` method produces a similar result to the `beforeShow` property that we looked at earlier. However, the method is called in a different way of course. We can rework the internationalized date picker example we looked at to make use of the `change` method instead of the `beforeShow` property. In a new file in your text editing tool, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <link rel="stylesheet" type="text/css" href="styles/
intlPicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 14</title>
  </head>
  <body>
    <a id="uk" href="#" title="English"></a><a id="france" href="#"
title="Français"></a>
    <div class="row"><label>Enter a date: </label><input id="date">
</div>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/i18n/
ui.datepicker-fr.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //create default english date picker
        $("#date").datepicker($.datepicker.regional['']);

        //define click handler for uk flag
        $("#uk").click(function() {

          //change date picker using default language as config object
          $("#date").datepicker("change", $.datepicker.regional['']);

        });

       //define click handler for french flag
        $("#france").click(function() {

          //change date picker using french config object
         $("#date").datepicker("change", $.datepicker.regional['fr']);
        });
     });
    </script>
  </body>
</html>
```

Save this as `datePicker14.html`. This file differs from example 8 in that each of the flag buttons have their own click handler. Within each click handler we use the `change` method to set the `regional` property of the manager object.

To make the date picker English by default, we set the `regional` property of the manager object within the constructor method. We could also make it French by default by simply not setting the `regional` property and letting the widget pick up the language pack automatically.

As you can see, using the `change` method is extremely easy. Whenever one of the flag icons is clicked, the `change` method is called and the appropriate method is passed into the constructor method as an argument.

It works exactly as intended, although once the date picker has been opened the first time, clicking an icon will instantly open the date picker. Perhaps we could do with a `suppressDisplay` property to prevent this happening!

# Putting the date picker in a dialog

The `dialog` method produces the same highly usable and effective date picker widget, but displays it in a floating dialog box. The method is easy to use, but makes some aspects of using the widget non-autonomous, as we shall see. In a new file in your editor, add the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogDatePicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 15</title>
  </head>
  <body>
    <div id="row"><label>Enter a date: </label><input id="date">
<a id="invoke" title="Click to open Date Picker Dialog" href="#">
</a></div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //define click handler for date picker link
        $("#invoke").click(function() {
          //create the date picker in a dialog
          $("#date").datepicker("dialog", "", updateDate);
          //write chosen date to input
          function updateDate(date) {
            $("#date").val(date);
          }
        });
      });
    </script>
  </body>
</html>
```

Save this as `datePicker15.html`. We still use the `datePicker` constructor method, but this time it is wrapped in a click-handler for the link we are using as the trigger button. The `dialog` method takes two arguments. The first argument can accept a string which is used to set the initial date of the date picker. In this example, we've supplied an empty string so the date picker defaults to the current date. The second argument is a callback function to execute when a date is selected.

This function automatically receives the selected date as an argument so it is easy to update the value of the input field using the callback. Updating the text field is something that we need to do manually following the use of the `dialog` method.

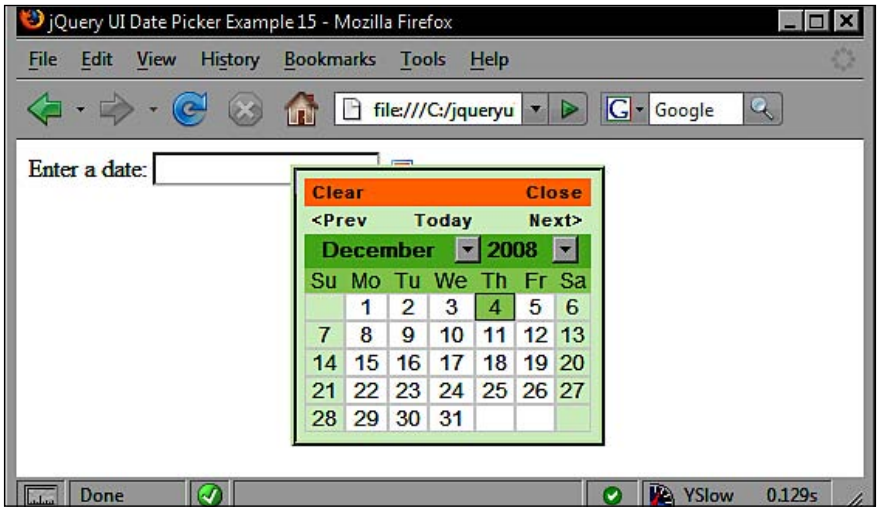
The reason we need to wrap the constructor method in the click handler, and use a button to open the date picker, is that if we don't do this, the dialog will automatically be displayed when the page loads and not when the `<input>` is focused.

A basic stylesheet is used to position the button, which is as follows:

```
#row { width:249px; }
#invoke {
  background:url(../img/date-picker/cal.png) no-repeat;
  width:16px; height:16px;
  display:block; float:right;
  margin-top:-18px;
}
```

This can be saved as `dialogDatePicker.css` in the `styles` folder.

Now when the date picker is opened, it will appear at the center of the page, floating above any existing content. The date picker has some additional styling, but essentially functions in the same way as before. The following screenshot shows how it appears:



The `destroy` method is used in the same way here as it is with the other widgets we have looked at, and the `enable`, `disable`, `isDisabled`, `show`, and `hide` methods are all intuitive and easy to use. Let's just quickly take a look at a generalized example that covers all of them. Create a new file in your text editor and add to it the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <link rel="stylesheet" type="text/css" href="styles/
dialogDatePicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Date Picker Example 16</title>
  </head>
  <body>
    <h2>Would you like to select a date using the date picker?</h2>
    <label for="yes">Yes</label><input type="radio" name="pickerGroup"
id="yes"><label for="notNow">Not now</label><input type="radio"
name="pickerGroup" id="notNow"><label for="notEver">Not ever
</label><input type="radio" name="pickerGroup" id="notEver"><br>
    <label>Enter a date: </label><input id="date">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //define config object
        var pickerOpts = {
          onSelect: handleSelect,
          beforeShow: handleShow
        };
        //create basic date picker
        $("#date").datepicker(pickerOpts);
        //add click handler for yes radio
        $("#yes").click(function(){
```

```
          //enable date picker and input if they are currently
disabled
          ($("#date").datepicker("isDisabled") == true) ? $("#date")
.datepicker("enable") : null;
        });
        //add click handler for not now radio
        $("#notNow").click(function(){
         //disable date picker and input if they are not currently
disabled
         ($("#date").datepicker("isDisabled") == false) ? $("#date")
.datepicker("disable") : null;
       });
      //add click handler for not ever radio
       $("#notEver").click(function() {
         //destroy date picker
         $("#date").datepicker("destroy");
         (!$("#date").attr("disabled")) ? null : $("#date")
.removeAttr("disabled") ;
         //add prompt text after input
         $("<p>").text("You will need to enter a date manually")
.insertAfter("#date");
       });
       //define handleSelect function
       function handleSelect(date) {
         //add prompt text after input
         $("<p>").text("You chose " + date + " is this correct?")
.attr("id", "prompt").insertAfter("#date");
         //add buttons
         $("<button>").text("Yes").attr("id", "btnYes")
.insertAfter("#prompt");
         $("<button>").text("No").attr("id", "btnNo")
.insertAfter("#btnYes");
         //add click handler for yes button
         $("#btnYes").click(function() {
           //change prompt text
            $("#prompt").text("Thanks for choosing a date!");
            $("button").remove();
          });
          //add click handler for no button
          $("#btnNo").click(function() {
            //reopen date picker
```

```
            $("#date").datepicker("show", "fast");
           $("#prompt").remove();
            $("button").remove();
          });
        }
        //define handleShow function
        function handleShow() {
          //add a close rollover
          $("<a>").text("rollover to close date picker").attr({
            id: "rollClose",
            href: "#"
          }).css({
            marginTop: "200px",
            display: "block"
          }).insertAfter("#date");
          //add rollover handler
          $("#rollClose").mouseover(function() {
            //close date picker
            $("#date").datepicker("hide", "fast");
            //remove rollover link
            $("#rollClose").remove();
             });
        }
      });
    </script>
  </body>
</html>
```

Save this as `datePicker16.html`. It's a huge page, for which I apologize, but it exposes a great deal of the method functionality that the widget has within it. The page is simple enough. We have a heading, some radio buttons, and the required `<input>` element. The final `<script>` block however is another story. Let's break down what each part does.

We first define the `pickerOpts` literal object and add function names as the values of the `onSelect` and `beforeShow` properties. As you know, these properties are called at different points during a date picker interaction. They are called when a date is selected and directly before the date picker is shown. We also associate the date picker with the `<input>` element in the usual way:

```
//define config object
var pickerOpts = {
  onSelect: handleSelect,
```

```
  beforeShow: handleShow
};

//create basic date picker
$("#date").datepicker(pickerOpts);
```

Next, we add the functionality for our radio buttons which allows us to make use of the `isDisabled`, `enable`, and `disable` methods. When the `yes` radio button is selected, we check whether the input or date picker are disabled and if so, call the `enable` method. If the `no` radio is selected, we do the opposite, after checking whether the input or date picker is enabled:

```
//add click handler for yes radio
$("#yes").click(function(){

  //enable date picker and input if they are currently disabled
  ($("#date").datepicker("isDisabled") == true) ? $("#date").
datepicker("enable") : null;
});

//add click handler for not now radio
$("#notNow").click(function(){

  //disable date picker and input if they are not currently disabled
  ($("#date").datepicker("isDisabled") == false) ? $("#date").
datepicker("disable") : null;
});
```

If the final radio button is selected, we use the `destroy` method to completely remove the date picker. We check whether the `<input>` element is disabled and if it is, we remove the `disabled` attribute so that a date can be entered manually. We also add some text to say that a date will now need to be entered manually.

```
//add click handler for not ever radio
$("#notEver").click(function() {

  //destroy date picker
  $("#date").datepicker("destroy");

  (!$("#date").attr("disabled")) ? null : $("#date").
removeAttr("disabled") ;

  //add prompt text after input
  $("<p>").text("You will need to enter a date manually").
insertAfter("#date");
});
```

Next, we need to add the callback functions which were supplied as values for the two properties in our configuration object. The `handleSelect` function, which will be executed every time a date is selected, adds some text to the page verifying that the selected date is correct. If the `no` button is clicked, we use the `show` method to reopen the date picker. We can also control the speed at which the date picker is displayed by supplying a second argument (the string `fast` in this case):

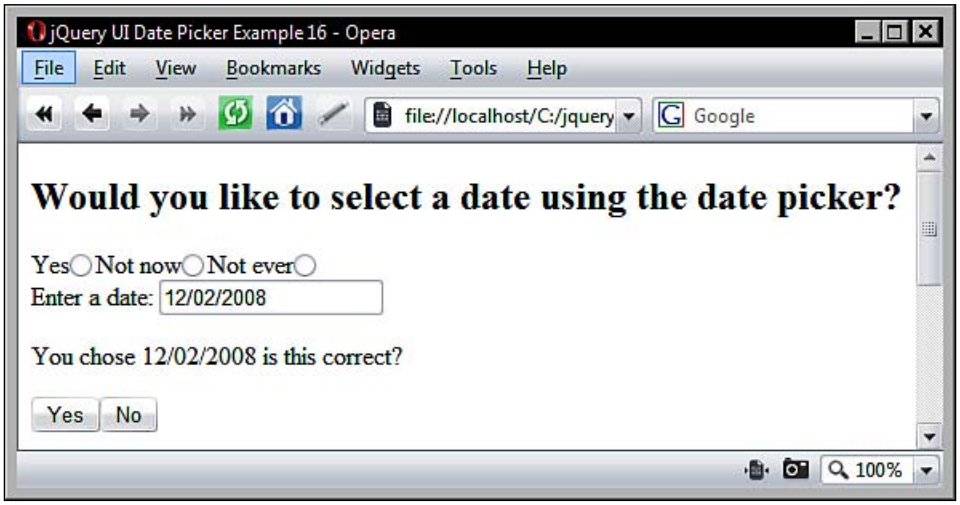
```
//define handleSelect function
function handleSelect(date) {
  //add prompt text after input
  $("<p>").text("You chose " + date + " is this correct?").attr("id",
"prompt").insertAfter("#date");
  //add buttons
  $("<button>").text("Yes").attr("id", "btnYes").
insertAfter("#prompt");
  $("<button>").text("No").attr("id", "btnNo").insertAfter("#btnYes");
  //add click handler for yes button
  $("#btnYes").click(function() {
    //change prompt text
    $("#prompt").text("Thanks for choosing a date!");
    $("button").remove();
  });
  //add click handler for no button
  $("#btnNo").click(function() {
    //reopen date picker
    $("#date").datepicker("show", "fast");
    $("#prompt").remove();
    $("button").remove();
  });
}
```

Finally, we add the callback function for the `beforeShow` property. With this function we add a link to the page which, calls the `hide` method to close the date picker without selecting a date when rolled over:

```
//define handleShow function
function handleShow() {
  //add a close rollover
  $("<a>").text("rollover to close date picker").attr({
    id: "rollClose",
    href: "#"
  }).css({
    marginTop: "200px",
```

```
      display: "block"
    }).insertAfter("#date");

    //add rollover handler
    $("#rollClose").mouseover(function() {

      //close date picker
      $("#date").datepicker("hide", "fast");

      //remove rollover link
      $("#rollClose").remove();
    });
  }
```

Using a rollover in this example is necessary as clicking outside of the date picker while it is open closes it automatically. Here's a screenshot of how the page should look after a date has been selected:




# Fun with date picker

We've nearly looked at all of the inherent functionality of the date picker. I'm not saying we've covered everything that can be done with it of course, but we've looked at enough now to have a very good understanding of the properties and methods that it exposes. However, I've saved a couple of properties and methods for us to have some fun with in our last date picker example.

For this example, we'll work a little AJAX magic into the mix and create a date picker, which prior to opening, will communicate with the server to see if there are any dates that cannot be selected. It could be a date picker that a freelance consultant uses to accept bookings from clients. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.datepicker.css">
    <link rel="stylesheet" type="text/css" href="styles/
ajaxDatepicker.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI AJAX Date Picker</title>
  </head>
  <body>
    <div class="container">
      <p>Use the date picker to request a consultation period</p>
      <label>Book period:</label><input id="date">
      <div class="key"></div><label class="keyLabel"> = Already
Booked</label>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.datepicker.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){

        //arrays to hold ajax dates
        var months = [];
        var days = [];

        //get pre-booked dates from server
        function getDates() {

   $.getJSON("http://www.danwellman.co.uk/
bookedDates.php?jsoncallback=?",

          function(data) {

            //process results
```

```
            for (x = 0; x < data.dates.length; x++) {
              //put results into arrays
              months.push(data.dates[x].month);
              days.push(data.dates[x].day);
            }
          });
        };
        getDates();

        //define config object
        var pickerOpts = {
         beforeShowDay: addDates,
         minDate: +1
        };

        //create date picker
       $("#date").datepicker(pickerOpts);

        //add pre-booked dates to datepicker
        function addDates(date){

          //filter out weekends
          if (date.getDay() == 0 || date.getDay() == 6) {
            return [false, "weekend_class"];
          }

          //check each day in arrays
          for (x = 0; x < days.length; x++) {
            //if date is same as current day...
            if (date.getMonth() == months[x] - 1 && date.getDate() ==
days[x]) {
              //make day unselectable
              return [false, "preBooked_class"];
            }
          }

          //other dates are selectable
          return [true, ''];
        }    });
    </script>
  </body>
</html>
```

Save this as `AJAXdatepicker.html`. The first part of our script initially declares two empty arrays, and then creates a new function called `getDates`. This function performs an AJAX request and obtains the JSON object from the PHP file.

The format of the JSON is an object called `dates` which contains an array of objects. Each object contains a `month` and a `day` property representing one date that should be made unselectable. The function pushes the value of each property into either the `months` or `days` array for a later function's use. We immediately call this function so the results can already be processed for the date picker to use.

```
//arrays to hold ajax dates
var months = [];
var days = [];

//get pre-booked dates from server
function getDates() {

      $.getJSON("http://www.danwellman.co.uk/
bookedDates.php?jsoncallback=?",

  function(data) {

    //process results
    for (x = 0; x < data.dates.length; x++) {

      //put results into arrays
      months.push(data.dates[x].month);
      days.push(data.dates[x].day);
    }
  });
};
getDates();
```

Next, we define a configuration object for the date picker. The properties of the object are simply the callback function to make the dates specified in the JSON object unselectable, and the `minDate` property will be set to the relative value `+1` as we don't want people to book dates in the past, or on the current day:

```
//define config object
var pickerOpts = {
  beforeShowDay: addDates,
  minDate: +1,
};
```

Finally, we define the `addDates` callback function which is invoked on the `beforeShowDay` event. This event occurs once for each of the 35 individual day squares in the date picker. Even the empty squares.

Our function is passed the current date and must return an array containing two items. The first is a boolean indicating whether the day is selectable, and optionally a class name to give the date.

Our function first checks to see whether the day portion of the current date is equal to either `0` (for Sunday) or `6` (for Saturday). If it is, we return `false` as the first item in the array, and specify the class name `weekend_class` as the second item.



There is a built-in function of the manager object, `$.datepicker.noWeekends`, which automatically makes weekends unselectable. This is specified as the value of the `beforeShowDay` property when used, but we cannot use it in this example as we are providing our own callback function.

We then loop through each item in our `months` and `days` arrays to see if any of the dates passed to the callback function match the items in the arrays. If both the `month` and `day` items match a date, the array returns with `false` and a custom class name as its items. If the date does not match, we return an array containing `true` to indicate that the day is selectable. This allows us to specify any number of dates that cannot be selected in the date picker:

```
//add pre-booked dates to datepicker
function addDates(date){
  //filter out weekends
  if (date.getDay() == 0 || date.getDay() == 6) {
    return [false, "weekend_class"];
  }
  //check each day in arrays
  for (x = 0; x < days.length; x++) {
    //if date is same as current day...
    if (date.getMonth() == months[x] - 1 && date.getDate() == days[x])
{
      //make day unselectable
      return [false, "preBooked_class"];
    }
  }
  //other dates are selectable
  return [true, ''];
}
```

In addition to the HTML page, we'll also need a little styling. In a new page in your editor, create the following stylesheet:

```
.container {
  border:1px solid #3399cc; width:380px;
  height:250px; padding:0 0 0 10px;
}
p {
```

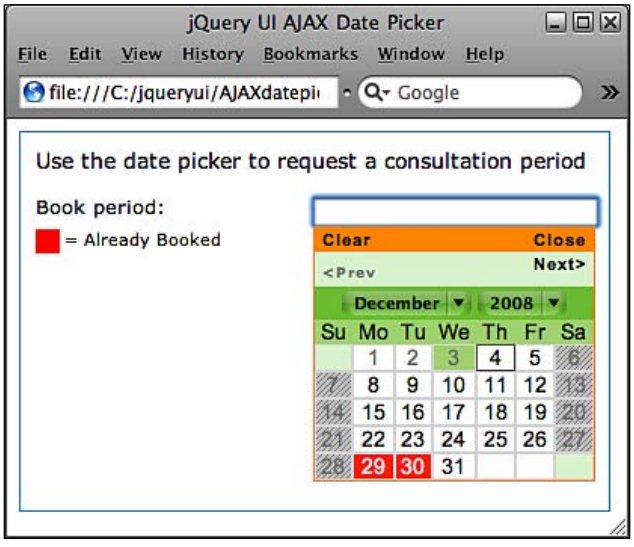
```
    font-family:Verdana; font-size:90%;
    margin-top:10px;
  }
  label {
    margin-right:98px; float:left;
    font-family:Verdana; font-size:80%;
  }
  .keyLabel {
    font-size:70%;
  }
  .key {
    width:16px; height:16px;
    background-color:#FF0000;
    float:left; margin-right:3px;
  }
  .divider {
    background:url(../date-picker/img/divider.gif) repeat-x;
    width:160px; height:2px; clear:both;
    position:relative; top:10px;
  }
  #date {
    width:183px; text-align:center;
  }
  .datepicker .weekend_class {
    background:url(../img/date-picker/weekendRepeat.gif) repeat-x;
  }
  .datepicker .preBooked_class {
    color:#fff; background-color:#FF3300;
  }
  #confirmation {
    font-size:70%; width:150px;
    line-height:2em; position:relative;
    top:10px;
  }
  #request {
    margin:10px 0 0 10px;
  }
  #thanks {
    font-size:70%; margin-top:15px;
  }
```

Save this as `AJAXdatepicker.css` in your `styles` folder. I used PHP to provide the JSON object in response to the AJAX request made by our page. If you don't want to install and configure PHP on your web server, you can use the file that I have placed at the URL specified in the example. For anyone that is interested, the PHP used is as follows:

```
<?php
  header('Content-type: application/json');
  //define booked dates as JSON
  $dates = "({
    'dates':[
      {'month':12,'day':2},
      {'month':12,'day':3},
      etc...
  ]})";
  $response = $_GET["jsoncallback"] . $dates;
  echo $response;
?>
```

The pre-booked dates are just hard-coded into the PHP file. Again, in a proper implementation, you'd probably need a more robust way of storing these dates, such as a database.

When you run the page in your browser and open the date picker, the dates specified by the PHP file should be styled according to our class name and should also be completely non-responsive, as in the following screenshot:



# Summary

We looked at the date picker widget in this chapter, which is supported by one of the biggest APIs in the jQuery UI library. This gives us a huge number of properties to work with and methods to receive data from.

We looked at the default implementation and how much behavior is added to the widget automatically. We then moved on to look at the different ways in which a new theme for the widget can be created.

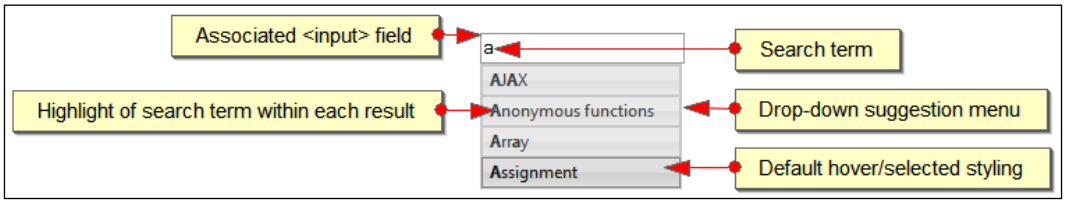
We also looked at how easy the widget makes implementing internationalization. We saw that there are thirty four additional languages the widget has been translated into, each of these are packed into a module that is easy to use in conjunction with the date picker for adding support for alternative languages. We also saw how we create our own custom language configuration.

We covered some of the events that are fired during a date picker interaction, and looked at the range of methods available for working with and controlling the date picker from our code. It's been a long chapter because of the amount of functionality that is wrapped up in this fantastic component.

# 7
# Auto-Complete

The auto-complete text field is a popular Web 2.0 function that is being seen on more and more sites thanks to its inclusion in some of the best JavaScript libraries around. A recent addition to jQuery UI, it exposes an API that is richly populated with useful methods and properties to make this widget work in the best possible way.

For anyone that hasn't experienced an auto-complete, its basic functionality is simple. It is attached to a standard text `<input>` field and when a visitor begins typing in the text field, a menu drops down showing suggestions starting with the letters that have been typed in the field. The following image illustrates this:




Throughout this chapter, we'll be investigating the following aspects of the widget:

- Basic implementation
- Auto-complete styling
- Configuring auto-complete properties
- Local and remote data
- Reacting to change with auto-complete callbacks
- Working with auto-complete's methods

The underlying structure of the auto-complete menu is based on a simple and semantic unordered list element, where each suggestion within the menu is added as a single `<li>` element.

# Basic implementation

Let's put the widget into practice so we can see just how easy it is to use it on the page in a fully functional, default implementation. Let's start off with the following new HTML page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 1</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
    jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
    ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
    ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
          "Scope", "Typeof", "Undefined", "Variables", "XHR"
        ];

        //create config object
        var autocompOpts = {
          data: suggestions
```

```
        }
        //turn specified element into an auto complete
        $("#suggest").autocomplete(autocompOpts);
      });
    </script>
  </body>
</html>
```

Save the page as `autocomplete1.html`. The page consists of just a simple `<label>` and the text `<input>` that our auto-complete is to be associated with. The `<script>` is almost as simple too. We define a literal array of the values we want to appear in the suggestion menu, and a configuration object with a single key, the `data` property. This property is given the name of the local data source, and the object is then passed to the constructor method as an argument.

The `autocomplete` constructor method is called using a jQuery object representing the associated `<input>`. It really is that simple. When you run this page in your browser and begin typing in the text field, you should see the suggestion menu shown in the screenshot at the start of this chapter.

Because auto-complete is such a new component, at the time of writing it doesn't yet have its own `flora` theming. It does have some basic styles applied to it using the `default.all.css` stylesheet however, so we can use this instead of `flora`. Our auto-complete example will look the same as the first screenshot of this chapter.

As I mentioned, we used an array literal in this example for our suggestion entries. We could also supply an array generated by the return of another function, such as the result of JavaScript's native `split()` method. Several library files are required for the auto-complete widget to function:

- `ui.all.css`
- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.autocomplete.js`

# Configurable properties

The auto-complete widget includes a wide range of configurable properties that can expose different behaviours depending on your specific requirements. These properties are laid out in the following table:

| Property | Default | Usage |
|---|---|---|
| `autoFill` | `false` | Adds the first suggestion to the text `<input>` automatically when the *Enter* or *Tab* key is pressed |
| `cacheLength` | `10` | Configures the number of entries stored in the local cache when using a remote data source and can be disabled by setting `cacheLength` to **1** |
| `data` | | Links the widget to the local data source |
| `delay` | `10 (400 for remote)` | Specifies the number of milliseconds after the visitor begins typing in the text field before the suggestion menu is displayed |
| `extraParams` | | Specifies extra parameters that are appended to the request URL and passed to the server when a query to the remote data source is made |
| `formatItem` | | Defines a function which is used to format how each item in the suggestion menu appears |
| `formatMatch` | | Defines a function which provides additional data that is added to the text field but not returned from the data source |
| `formatResult` | Plain text | Defines a function which provides formatting for the result that is placed in the text field |
| `highlight` | `true` | Defines a function which provides custom formatting of the highlight |
| `matchCase` | `false` | Enables case sensitivity on the auto-complete and should only be used with remote data in certain cache configurations |
| `matchContains` | `false` | Enables matching within each result string instead of just at the start of it |
| `matchSubset` | `true` | Enables the local cache for more specific queries of previously returned results |

| Property | Default | Usage |
|---|---|---|
| max | 100 | Sets the maximum number of suggestions in the drop-down menu |
| minChars | 1 | Sets the number of characters that must be entered into the <input> before the suggestion menu appears |
| multiple | false | Enables the selection of multiple suggestions |
| multipleSeperator | "," | Defines the separator for selected suggestions that are placed in <input>, or <textarea> |
| mustMatch | false | Specifies that a suggestion from the menu must be selected |
| scroll | true | Automatically scrolls the suggestion menu when there are too many results to fit into the menu |
| scrollHeight | 180 | Sets the height of the suggestion menu when scroll is set to true |
| selectFirst | true | Automatically selects the first item in the suggestion list when the *Tab* or *return* key is hit |
| width | width of the <input> | Sets the width of the suggestion menu |
| url | | Specifies the URL of the remote data store |

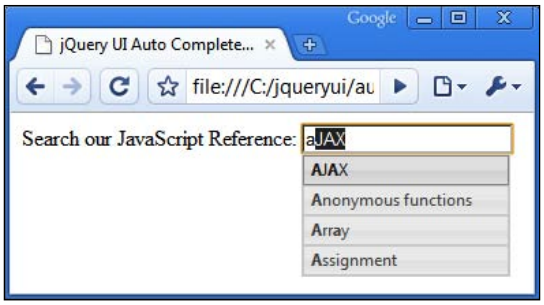
Let's look at what effect some of these properties have on our auto-complete instance. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
    themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 2</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
    jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/ui.core.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
          "Scope", "Typeof", "Undefined", "Variables", "XHR"
    ];

        //create config object
        var autocompOpts = {
          data: suggestions,
          autoFill: true
        };

        //turn specified element into an auto-complete
        $("#suggest").autocomplete(autocompOpts);
      });
    </script>
  </body>
</html>
```

Save this as `autocomplete2.html`. We use a literal object to configure the `autoFill` property. This configuration object is then passed to the auto-complete constructor function as the first (and in this example, the only) argument.

The `autoFill` property automatically inserts the first suggestion into the associated text field. It selects the text that has been added so that it gets removed if the visitor continues typing or selects another value. The following screenshot shows this feature in action:

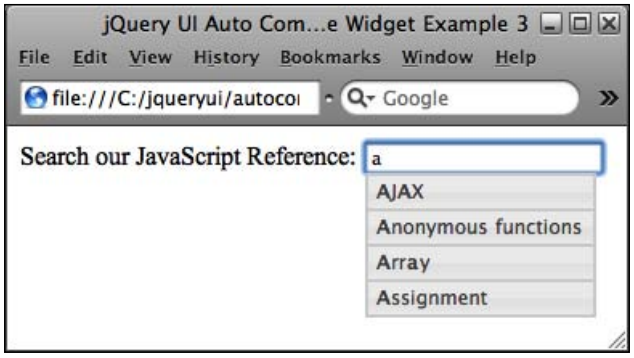



Next, we can put some of the other properties to work. Change the configuration object used in `autocomplete2.html` so that it appears as follows:

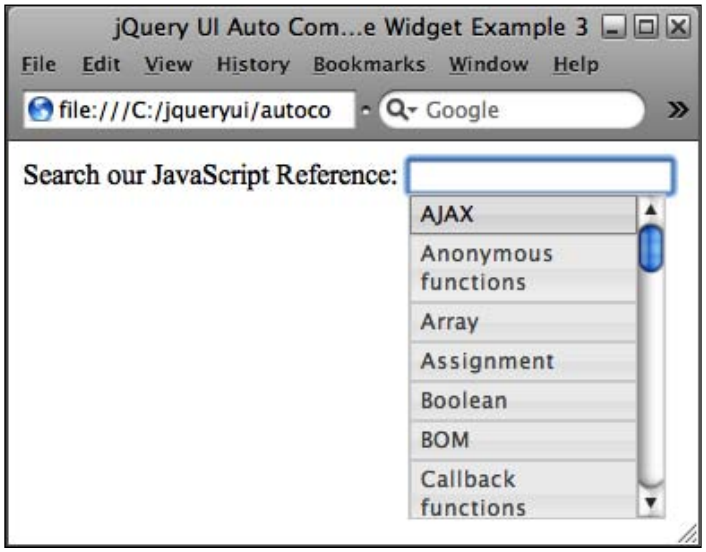
```
//create config object
var autocompOpts = {
  data: suggestions,
  selectFirst: false,
  minChars: 0
};
```

Save this variant as `autocomplete3.html`. By default, when the auto-complete suggestion menu opens, the first suggestion, or match, is selected in the menu. Pressing the *Enter* or *Tab* key on the keyboard will copy the suggestion to the text field.

Setting the `selectFirst` property to `false` disables this behavior. Therefore, when the suggestion menu initially opens, none of the matches are selected and pressing *Enter* or *Tab* does nothing. The next screenshot shows the suggestion menu as it appears when it initially opens with nothing selected:



Setting the `minChars` property to `0` also adds an additional feature. When you start typing in the text field, the suggestion menu appears with the matching data as expected. But if you remove what you've typed in the text field, the menu remains and displays all of the data (up to the number configured in the `max` property, which is `100` by default) from the data source. This is illustrated in the following screenshot:




# Scrolling

In the last example, we saw that the height of the suggestion menu had a fixed maximum size. When there were just three or four results shown in the menu, it accommodated all of them quite easily, but when all of the data was returned, the menu did not endlessly increase in height to show them all at once.

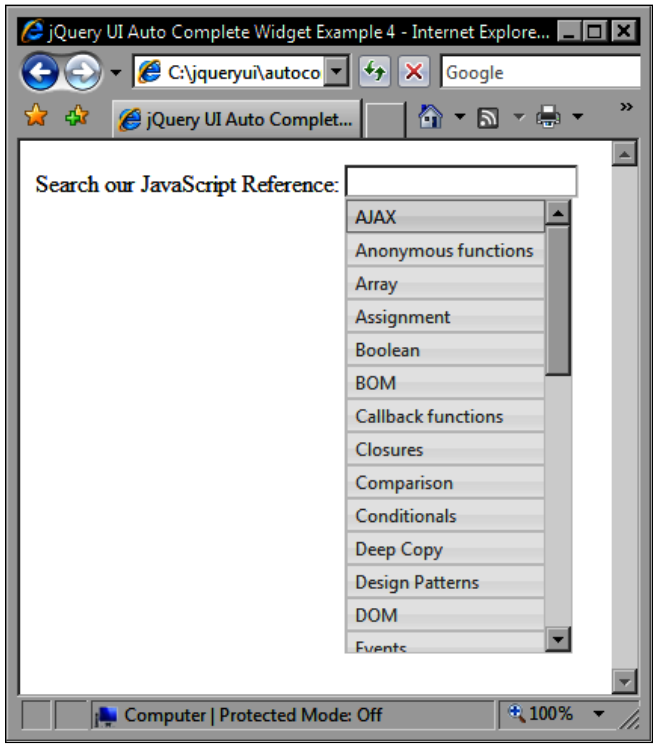
This maximum size is controlled via the `scrollHeight` property, which is used in conjunction with the `scroll` property. The `scrollHeight` property takes an integer, which represents the maximum number of pixels that the menu should grow, before a vertical scroll bar is added.

The `scroll` property is a boolean which enables or disables scrolling, and therefore nullifies the `scrollHeight` property. With `scroll` set to `false`, the menu will grow indefinitely until it is showing the configured maximum number of results.

If you're working with a large data set, and there are likely to be many results to display, increasing the height of the suggestion menu can allow you to display more results at once, without letting the menu get too large. To see this property at work, change the configuration object in `autocomplete3.html` to the following:

```
//create config object
var autocompOpts = {
  data: suggestions,
  minChars: 0,
  scrollHeight: 300
};
```

This revision can be saved as `autocomplete4.html`. We still specified `0` for the value of the `minChars` property because our dataset only has a maximum of about 4 results for any one character. We would need to generate a lot of results for the `scroll` feature to become noticeable. By setting the value of the `scrollHeight` property to `300`, we double the height of the default auto-complete menu, as shown in the following screenshot:



# Auto-complete styling

The default styling of auto-complete provides a serious and professional, if somewhat basic, appearance for the widget. Changing any of the different elements' style is as easy as overriding any style rules that you choose. To make things a little easier for you, several properties are provided that deal with aspects of the widget's visual appearance. Let's look at these next. Change `autocomplete4.html` so that it resembles the following:

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <link rel="stylesheet" type="text/css" href="styles/
myAutocomplete.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 5</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
          "Scope", "Typeof", "Undefined", "Variables", "XHR"
```

```
        ];
        //create config object
        var autocompOpts = {
          data: suggestions,
          width: 150
        };
        //turn specified element into an auto complete
        $("#suggest").autocomplete(autocompOpts);
      });
    </script>
  </body>
</html>
```

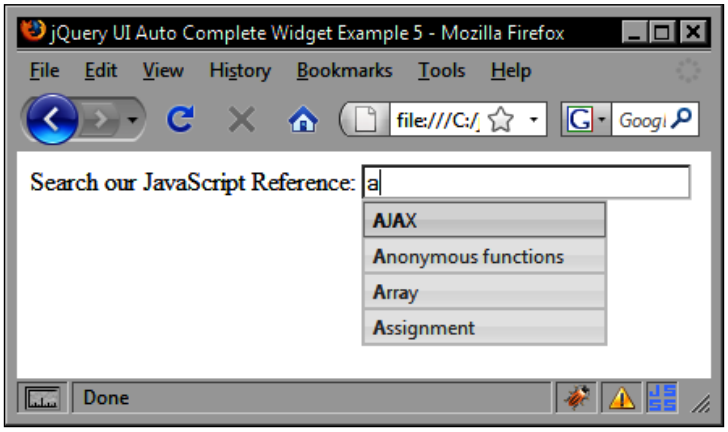
Save this as `autocomplete5.html`. An additional, and very basic, stylesheet is required for this example. All that is needed is to do is set the width of the `<input>` to `200px`:

```
#suggest { width:200px; }
```

By default, the width of the auto-complete drop-down suggestion menu is the same as the width of the `<input>` element it is associated with. The `<input>` will always appear slightly longer, however, due to padding and borders, which can be altered if required.

So, if we set the width of the `<input>` to `200px`, as we did in this example, by default, the suggestion menu will also appear this long. However, the `width` property of auto-complete, specified as an integer representing a pixel value, will override this. In this example, the menu will appear substantially thinner than the text field.

Now the example page should appear as in the following screenshot:



Another property related to the visual appearance and the structural format of the suggestion menu is the `highlight` property. We can supply a function as the value of this property which can be used to customize the appearance of the highlight effect.

This is the effect that makes the matched search term in each suggestion appear in bold, in the previous screenshot, the first letter of each word was highlighted. The default `highlight` function wraps the search term in `<strong>` tags to give it a bold appearance. Let's change this so that it displays an image instead. Change `autocomplete5.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <link rel="stylesheet" type="text/css" href="styles/
customHighlight.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 6</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>

    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
```

```
            "Scope", "Typeof", "Undefined", "Variables", "XHR"
        ];

            //function to provide custom highlighting
            var customHighlight = function(val, term) {

              //replace search term with image
              return val.replace(new RegExp("(?![^&;]+;)(?!<[^<>]*)(" +
term.replace(/([\^\$\(\)\[\]\{\}\*\.\+\?\|\\])/gi, "\\$1") + ")(?![^<>
]*>)(?![^&;]+;)", "gi"), "<img src='img/letters/$1.png' alt='$1' />");

              //return "<img src='img/letters/$1.png' alt='$1' />"
             };

            //create config object
            var autocompOpts = {
              data: suggestions,
              highlight: customHighlight
            };

            //turn specified element into an auto-complete
            $("#suggest").autocomplete(suggestions, autocompOpts);
          });
      </script>
    </body>
</html>
```

Save this as `autocomplete6.html`. We've provided the name of a callback function as the value of the highlight property. We also defined the function in a separate code block preceding the configuration object. The function itself is relatively simple and accepts each value returned from the data source, and the actual search term, as arguments.

The function will be executed for each matching value from the data source (up to the configured maximum number of results), and will receive a new result in the `value` argument each time it is called.

Inside the function is a regular expression that looks decidedly unfriendly. Even for those of us that have worked with regular expressions before, it looks considerably advanced. This line of code has been copied out of the library into our function for the simple fact that it works well and efficiently.

We cannot code values directly into our highlight function. Even for a small data set, such as ours, it would still be a massive duplication of code. A regular expression is the only way to look for, and respond to, particular patterns of characters without hard-coding anything.

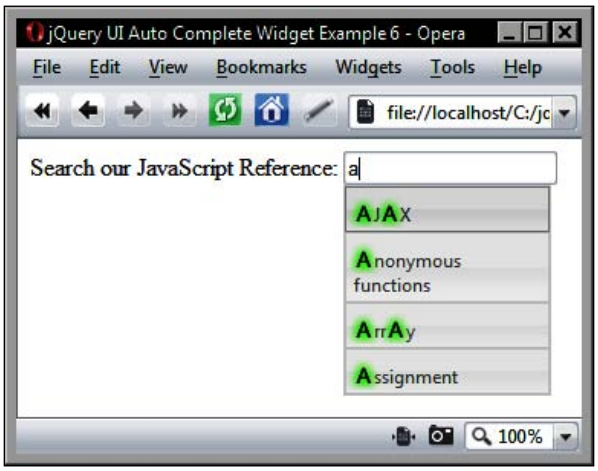
We could choose to write our own regular expression that matched the search term and replaced it with arbitrary HTML. This would undoubtedly look similar to the original expression used by the widget so we may as well just use the same code used in the widget and be done with it.

What we have done, however, is change the second argument of the outer `replace` method. So instead of wrapping the search term in `<strong>` tags, it instead creates an image, using the search term as the image filename and its `alt` text.

We need some more styling to make our images sit within each list item correctly. In a new stylesheet, add the following selectors and rules:

```
.ui-autocomplete-results li img {
  position:relative; top:4px; left:3px; margin-left:-8px;
}
```

This file can be saved as `customHighlight.css` in the `styles` folder. Now, provided we have the necessary images (one for each letter of the alphabet that our data source contains results for), the function will work as expected, as shown in the following screenshot:



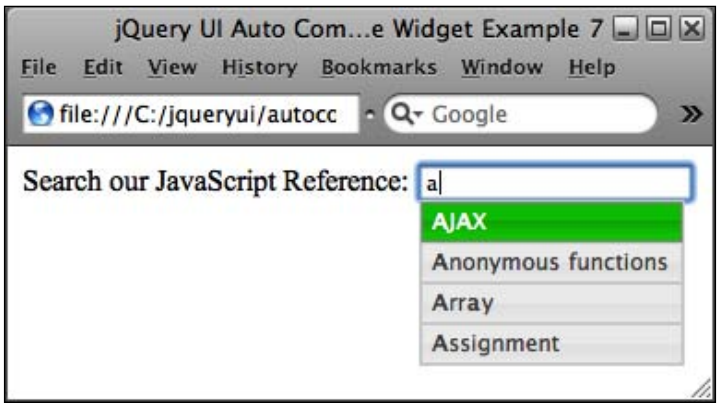

Highlighting can also be switched off altogether by supplying `false` as the value of the `highlight` property.

Apart from changing the appearance of the highlighted search term within each suggestion, we can also very easily change the hover and selected state used in the menu by overriding a single simple style rule.

The class name `ui-autocomplete-over` is automatically appended to whichever `<li>` element is currently selected. By default, this class provides a slight darkening of the border, text, and background image styling the element. If we wish to change this, we can simply provide our own style rules for the `ui-autocomplete-over` class name. Create a new file and add the following code to it:

```
.ui-autocomplete-results li.ui-autocomplete-over {
  background:url(../img/autoCompSlice.gif) repeat-x;
  color:#ffffff !important;
}
```

Save this stylesheet as `myAutocomplete2.css`. You'll also need to update the link in `autocomplete6.html` to point to this new CSS file and get rid of all the `customHighlight` code. Save the change as `autocomplete7.html`. The hover and selected states should now appear like this:




# Multiple selections

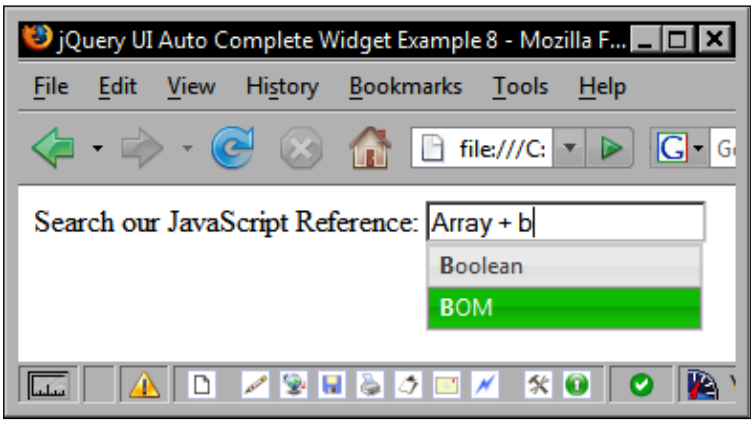
Apart from selecting single suggestions from the auto-complete menu, we can also configure the widget so that it allows multiple selections to occur. This will allow users of the widget to select several suggestions from the menu. There are two properties related to multiple selections:

- `multiple`
- `multipleSeparator`

Their usage is simple. The `multiple` property takes a boolean indicating whether or not to enable multiple selections, `multipleSeparator` allows us to specify an alternative character from the default comma to be used to separate the selected results. Change our configuration object to this:

```
//create config object
var autocompOpts = {
  data: suggestions,
  multiple: true,
  multipleSeparator: " + "
};
```

This variant can be saved as `autocomplete8.html`. Now when we view the page, we see that once we have made an initial selection from the suggestion menu, we can continue typing in the `<input>` to invoke another comparison of the data source and make another selection:




# Advanced formatting

Apart from altering the appearance of the search term highlight and the hover and selected states, several properties are provided by the auto-complete API that lets us provide advanced formatting for several other aspects of the widget. The items that can be changed are:

- Individual items that appear as `<li>` elements in the menu
- The selected item that is added to the associated `<input>`
- New items to be added to the `<input>` when a match is selected that hasn't come from the data source

Like the `highlight` property that we looked at earlier, these aspects of the widget are configured using properties and take functions that should return the new mark-up that is to be used. The properties used for this advanced formatting are:

- `formatItem`
- `formatResult`
- `formatMatch`

Let's first look at configuring advanced formatting of the individual items in the list. Let's say, for example, that we wanted to include a link next to each suggestion in the drop-down menu. Change `autocomplete8.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 9</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
<script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
```

```
            "Prototype", "Regular Expressions", "Strings",
            "Scope", "Typeof", "Undefined", "Variables", "XHR"
          ];

          //provide additional markup for each result
          function customItem(data, i, max) {

            //add the link
            return data[0] + " (<a href='" + data[0] + "'.html' title='"
+ data[0] + "'>" + data[0] + "</a>)"
          }

          //create config object
          var autocompOpts = {
            formatItem: customItem
          };

          //turn specified element into an auto-complete
          $("#suggest").autocomplete(autocompOpts);
        });
      </script>
    </body>
  </html>
```

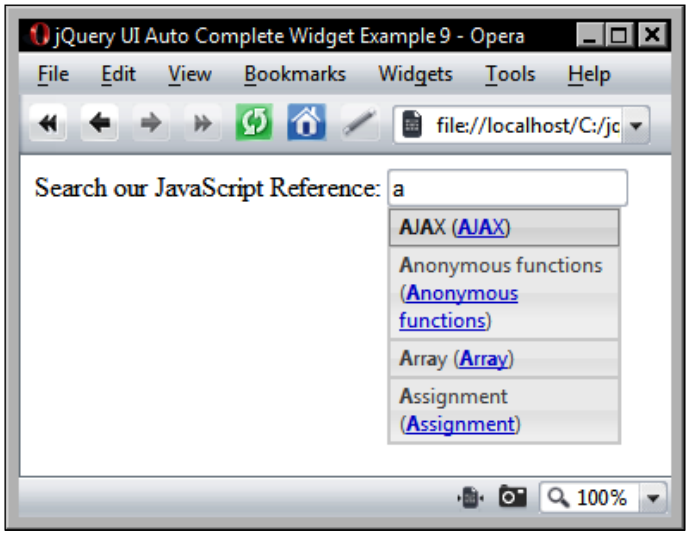
Save this file as `autocomplete9.html`. In this example, all we do is provide the additional mark-up, the `<a>` element within a literal string concatenated with the result from the data source for the `href` and `title` attributes.

Never mind that clicking on the link will cause the suggestion to be added to the text field instead of the link being followed, as we could easily add an additional click handler for this event and move the browser to the new page manually. Or, perhaps the links could be triggers for some kind of tooltip. The example is about adding additional formatting in the form of new mark-up, and that is exactly what we've done.

The `formatItem` property is useful as it allows us to specify an alternative format for the results presented in the suggestion menu. The premise is similar to that of the `highlight` function, although in the case of `formatItem`, a regular expression is not needed to achieve the custom formatting.

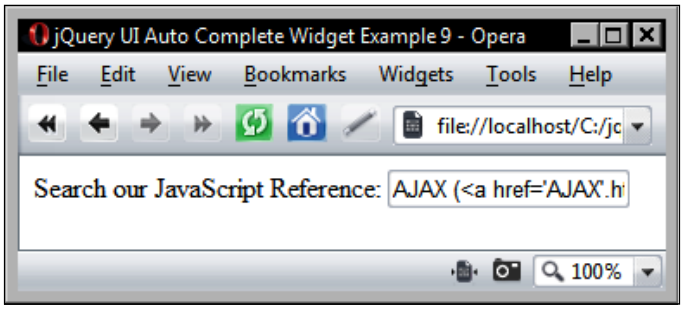
The function we specify as the value of `formatItem` receives four arguments when it is called. It receives the result, the row (or number) of the result, the total number of results, and the search term that was typed into the `<input>` field.

The `result`, is the data (the word returned as a result), and the `row` corresponds to the number of the result. What is meant by this is that the first result will have a row number of 1, the second result will have a row number of 2, etc. The `total` and `term` values should not need further explanation. The additional formatting provided by this example is shown in the following screenshot:




You should be aware of the fact that if the `formatItem` property is changed, the result that is added to the text field having been selected will automatically assume the formatting returned by `formatItem`.

This may not be the desired behavior as we may want just the result, not arbitrary data that we supply with `formatItem`, to appear in the text field. The next screenshot shows our text field when an item in the previous example has been selected:



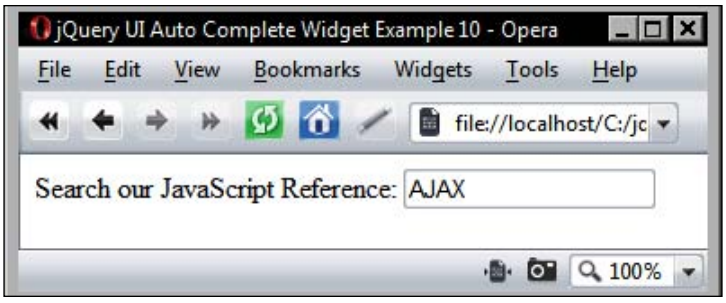
Thankfully, the auto-complete API provides the `formatResult` property which allows us to change the format of the data that is added to the text field following a selection. This can be most useful when working with custom formatting provided by `formatItem`. Following on from the previous example, change `autocomplete9.html` so that it appears thus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 10</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
          "Scope", "Typeof", "Undefined", "Variables", "XHR"
        ];
        //provide additional markup for each result
        function customItem(data, i, max) {
          //add the link
```

```
            return data[0] + "( <a href='" + data[0] + "'.html' title='"
+ data[0] + "'>" + data[0] + "</a> )"
          }
          //customize data added to text field
          function customResult(data, i, max) {
            //return just the data
            return data[0];
          }
          //create config object
          var autocompOpts = {
            data: suggestions,
            formatItem: customItem,
            formatResult: customResult
          };
          //turn specified element into an auto-complete
          $("#suggest").autocomplete(autocompOpts);
        });
      </script>
    </body>
  </html>
```

Save this version as `autocomplete10.html`. The function specified as the value of `formatResult` receives the same arguments as `formatItem`. To negate the custom formatting provided by the `formatItem` function, our `formatResult` function returns only the actual result. This removes the additional mark-up 'inherited' from the `formatItem` function.

The `<input>` field will now receive only the selected result, exactly as it did before we altered the formatting of the data within each `<li>` element in the suggestion menu, as shown in the following screenshot:




The `formatMatch` property allows us to add additional information to each result when it is added to the associated `<input>`.

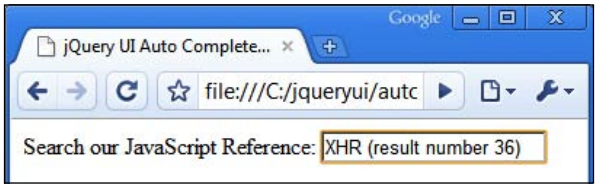
This additional data isn't part of the initial search of the data source, so it will not influence the returned matches in any way. It's similar in structure to the other format properties that we've looked at so far and receives the same arguments.

Change `autocomplete10.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 11</title>
  </head>
  <body>
    <label>Search our JavaScript Reference:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create local data
        var suggestions = [
          "AJAX", "Anonymous functions", "Array",
       "Assignment", "Boolean", "BOM",
          "Callback functions", "Closures", "Comparison",
          "Conditionals", "Deep Copy", "Design Patterns",
          "DOM", "Events", "Functions", "Global Variables",
          "HTML Output", "Inheritance", "JSON", "Keywords",
          "Logical Operators", "Loops", "Math", "Namespacing",
          "Null", "Objects", "Operators", "Properties",
          "Prototype", "Regular Expressions", "Strings",
          "Scope", "Typeof", "Undefined", "Variables", "XHR"
        ];
        //provide additional information to the selection that doesn't
get searched
        function customMatch(data, i, max) {
          return data + " (result number " + i + ")";
        }
        //define config object
```

```
        var autocompOpts = {
          data: suggestions,
          formatMatch: customMatch
        }
        //turn specified element into an auto-complete
        $("#suggest").autocomplete(autocompOpts);
      });
    </script>
  </body>
</html>
```

Save this as `autocomplete11.html`. All we do in this example is append some additional text to each selection when it is added to the text field. We use the `data` argument to return the actual selection, and the `i` argument to add which number the result is. After making a selection from the suggestion menu, the page should look like this:




We've been working with a flat data structure in our examples so far and have provided a simple array containing the data items to match search terms against. We can also use more complex objects within our data structure, with additional data supplied using arbitrary keys and values.

The keys will still be available to the functions for advanced formatting that we have just looked at. This will give us the ability to provide additional information alongside the matched results.

# Matching properties

There are several properties which are used to configure how matching occurs with the auto-complete search term and results. The `mustMatch` property, for example, configures the auto-complete so that the text field may only contain results supplied by the widget. Let's look at a basic example. Change the configuration object used with our auto-complete in `autocomplete12.html` to the following:

```
//create config object
var autocompOpts = {
  data: suggestions,
  mustMatch: true
};
```

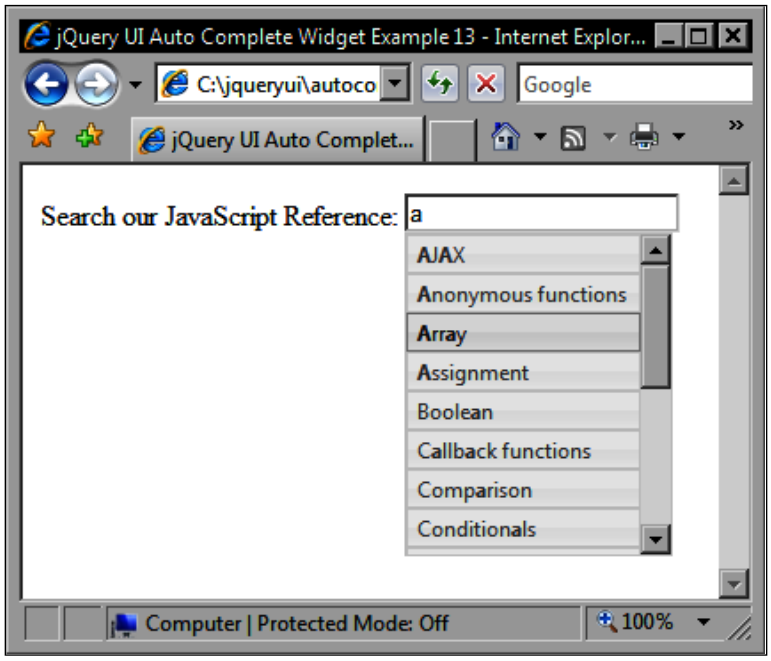
Save the change as `autocomplete12.html`. When we run this file in our browser, we see that as we start typing in the `<input>` with a letter that is a match for one of the results in our data set, the widget behaves as it has done so far.

If we start typing a letter in the `<input>` that isn't a match for our data (like a `z` for example) and then type another letter, the value of the `<input>` is cleared as a result of the `mustMatch` property.

Now let's look at a property that allows us to match not only the start of results in our data set, but to find matches within the strings which make up the data. Change the configuration object, used in the previous example, to this instead:

```
//create config object
var autocompOpts = {
  data: suggestions,
  matchContains: true
};
```

Save this variation as `autocomplete13.html`. Again, the premise with this property is simple. When a letter is entered into the text field, not just the results that start with the match are returned, but all results containing the match are returned as well, as you can see in the following screenshot:



# Remote data

We have used local data in all our examples so far, which is perfect for smaller data sets. The data will be cached once it has been loaded, and continue to be available to the text field while the page is open. It's efficient and easy to code.

However, when working with larger stores of data, it is more efficient to process the suggestions on the server and return only those suggestions that are required. Auto-complete makes working with remote data just as easy as working with local data, provided you have the back-end code to support it.

There are a series of properties that are used solely with remote data sets. These properties include the following:

- `cacheLength`
- `extraParams`
- `matchCase`
- `matchSubset`
- `url`

As we'll be working with remote data for the next few examples, we can use a slightly larger dataset. For this example, I've created a new MySQL database called `data` with a new table inside it called `countries`.

We'll be using the auto-complete widget to provide a list of countries of birth on what could be part of an account creation form. The data source we'll use for this example contains 128 records, which is still small enough to run locally with great efficiency, but is much larger than what we have worked with so far.

I usually use PHP where necessary because I'm somewhat used to and like its syntax. That's not to say that it's the only back-end language that could be used, or indeed the most practical, or most suited to the task at hand. Nevertheless, let's get started with the next example. In a new file in your text editor, add the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 14</title>
  </head>
```

```
  <body>
    <label>Please enter your country of birth:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create config object
        var autocompOpts = {
          url: "countries.php"
        }
        //turn specified element into an auto-complete
        $("#suggest").autocomplete(autocompOpts);
      });
    </script>
  </body>
</html>
```

Save this file as `autocomplete14.html`. Instead of using the `data` property to point to a local data source, we instead use the `url` property to point to the URL from which the data can be obtained. No additional configuration is required for this simple implementation, although we do, of course, need a backend.

For reference, the backend that I used for this example consists of the database I already mentioned and the following PHP file:

```
<?php
  /* connection information */
  $host = "localhost";
  $user = "root";
  $password = "your password here";
  $database = "autoComp";
  /* make connection */
  $server = mysql_connect($host, $user, $password);
  $connection = mysql_select_db($database, $server);
  /* get querystring parameter */
  $param = $_GET['q'];
  /* protect against sql injection */
```

```
  mysql_real_escape_string($param, $server);

  /* query the database */
  $query = mysql_query("SELECT * FROM countries WHERE country LIKE
'$param%'");

  /* loop through and return matching entries */
  for ($x = 0; $x < mysql_num_rows($query); $x++) {
    $row = mysql_fetch_assoc($query);
    $output = $row['country']."\n";
    echo $output;
  }

  mysql_close($server);
?>
```

I don't want to go into too much explanation here as this backend is just one of many possibilities and has been provided in supplementary information. I do want to mention the search term, which is passed to the backend as part of the GET request, and is available to our PHP file under the `q` super global variable.

We also included the `limit` request variable, which will be set to the same value as the configured `max` property. This property has the value of `100` by default with local data sources, but is set to `10` by the widget when using a remote data source.

The following screenshot shows the remote auto-complete in action. It looks exactly as it did before, but we know that our slightly larger data set is actually remote:

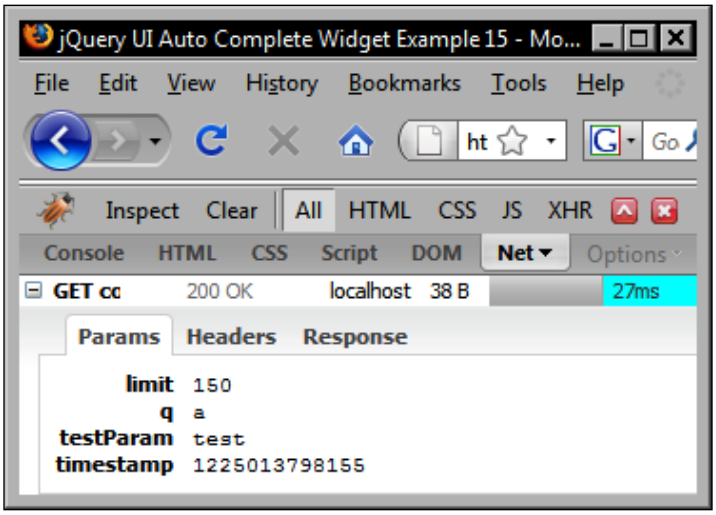


# Sending additional data to the server

The only remote configuration property not related to caching is the `extraParams` property. This can be used to send additional, arbitrary data to the server. Using this property is easy. All we need to do is provide a nested object literal as the value of the property. Change the configuration object used with `autocomplete14.html` to the following:

```
//create config object
var autocompOpts = {
  url: "countries.php",
  extraParams: {testParam:"test"}
};
```

Save this as `autocomplete15.html`. We're not actually going to do anything with the data, but using Firebug, we can easily see that our test data has been added to the query and sent with the rest of the request:




# Caching

The remaining remote properties are all related to the caching services provided by auto-complete, and they can all be used together for optimal cache performance. I mentioned earlier that a GET request is performed on every keystroke. That's not quite the whole truth and is where the local cache comes in.

Every time a keystroke is detected in the associated text field, the local cache is checked first for matching results, and then a GET request to the remote data source is made if nothing in the local cache matches.

The `cacheLength` property simply tells the widget how many items, if any, should be stored in the local cache. Caching can be disabled completely by setting this property to `1`.

The `matchCase` property configures the widget exactly the way that it implies. When this is set to `true`, the auto-complete widget becomes case sensitive, so the term **a** will not be a match for **A**.

Finally `matchSubset`, which by default is set to `true`, makes subsets of the items in the cache match, so when a second letter is typed into the input, only the subset of results from the cache will be shown.

# Auto-complete methods

Now that we've looked at the configurable properties supplied by the auto-complete API, let's move on to look at the methods it exposes. The following table lists the methods available when working with auto-complete:

| Method | Usage |
|---|---|
| `destroy` | Removes auto-complete functionality from the `<input>` |
| `flushCache` | Empties the cache |
| `result` | A function specified as the second argument to this method can be used to handle the selection of a result |
| `search` | Programmatically triggers the search event |
| `setData` | Supplies a new configuration object as the second argument of this method to update the configuration of the matched `<input>` element's auto-complete |

Like each of the other widgets, auto-complete has a method for removing auto-complete functionality programmatically. This is, of course, the `destroy` method that we have seen many times before. When called, the `<input>` element will no longer be associated with the auto-complete engine, and typing into it will not invoke the suggestion menu.

The `search` and `result` methods are closely linked. The `result` method allows us to specify a callback function that is triggered when the `search` event fires. This occurs either when a visitor selects a result from the suggestion menu, or when the `search` method is used to invoke the `search` event programmatically.

Let's look at the `result` method in our next example. Change `autocomplete15.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Widget Example 16</title>
  </head>
  <body>
    <label>Please enter your country of birth:</label>
    <input type="text" id="suggest">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create greets object
        var greets = {
          Albania: "Allo",
          Andorra: "Hola",
          Argentina: "Mari mari",
          Australia: "G'day Mate"
        }
        //create config object
        var autocompOpts = {
          url: "countries.php"
        }
        //turn specified elements into an auto-complete
        $("#suggest").autocomplete(autocompOpts);
        //set handler for selection of a result
        $("#suggest").autocomplete("result", function(event, data,
formatted) {
          (!$("#greeting")) ? null : $("#greeting").remove();
          $("<p>").text("You selected: " + formatted + ", " +
greets[data]).insertAfter("#suggest");
```

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

Save this file as `autocomplete16.html`. We set up a little `greets` object that holds localized greetings for all of the countries that begin with A. We should, of course, provide these for all of the countries in our table, but this would make the example far larger than it needs to be. We then create our configuration object and initialise the widget as normal.

Finally, we add an inline handler for the `result` event, and use it to retrieve the appropriate greeting from our `greets` object. It is then appended to the page whenever a suggestion from the menu is selected (after removing any previous messages if they exist):



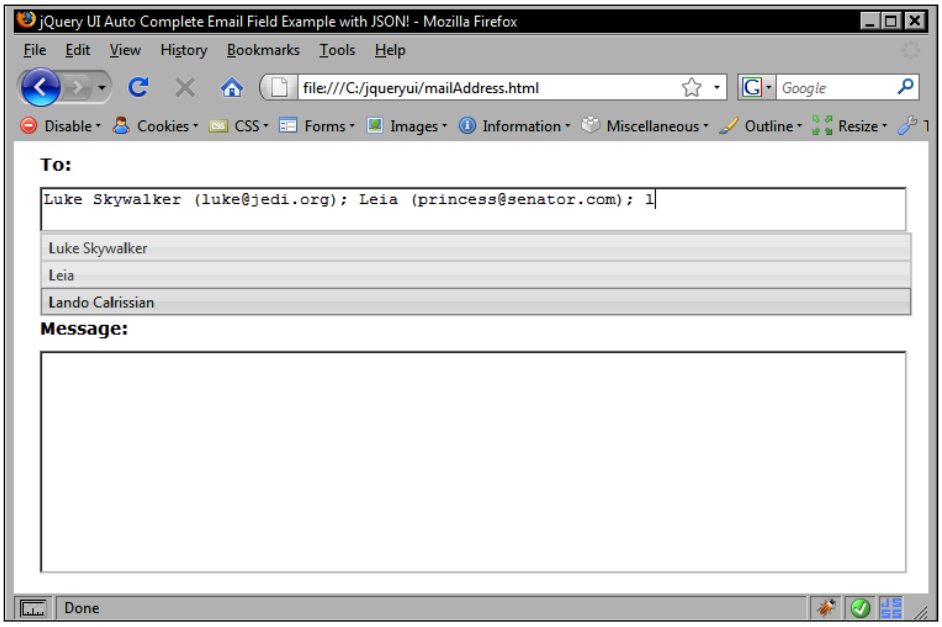

We looked at the local caching of back-end results that is handled by auto-complete and saw how this can be fine-tuned to suit an individual implementation. Apart from configuring the cache, the auto-complete API also gives us the means to empty the cache at any given point by calling the `flushCache` method.

Flushing the cache may be required during the execution of your application if there is a change to the data set that the auto-complete is tied to at the backend, and hence the URL used to query it.

Changing things like the URL that the widget gets its data from, or any of the other configuration properties, is made easy with the `setData` method. This method updates the configuration options of the auto-complete instance and takes as its second argument the new configuration object.

# Fun with auto-complete

For our last auto-complete example, we can create a simple email system front-end that features an auto-complete attached to an `<textarea>` that is used to enter the address(es) that the email will be sent to. It can be connected to a back-end data source containing the visitor's email contacts. The following screenshot shows the kind of result we're aiming for:




While we won't be using every property and method exposed by the API, we can certainly put a range of them to good use with this example to reinforce what we've learned over this the course of this chapter. Start off with the following basic web page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/default/ui.all.css">
    <link rel="stylesheet" type="text/css" href="styles/
mailAddress.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Auto Complete Email Field Example with JSON!
</title>
```

```
  </head>
  <body>
    <div id="mailContainer">
    <label>To:</label><textarea id="address" cols="10" rows="1">
</textarea>
    <label>Subject:</label><input type="text">
    <label>Message:</label><textarea cols="10" rows="10"></textarea>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.autocomplete.js"></script>
    <script type="text/javascript">
    </script>
  </body>
</html>
```

Save what we have so far as `mailAddress.html`. We're keeping the page simple for our final auto-complete example, using only the elements necessary to illustrate things. The first `<textarea>` will be the one that is associated with our auto-complete.

We'll need some basic CSS also while trying to keep this as simple as possible. Add the following rules to a new file:

```
#mailContainer { width:700px; margin:0 auto; }
label {
  display:block; width:700px;
  font:bold 14px Verdana, Arial, Helvetica, sans-serif;
  margin:10px 0;
}
input, textarea { display:block; width:700px; }
```

Save this as `mailAddress.css` in the `styles` folder. Now let's move swiftly on to the most important, and unquestionably the most fun part of the example, the JavaScript. In the empty `<script>` element in `mailAddress.html`, add the following code:

```
//function to execute when doc ready
$(function() {
  //get JSON
  $.getJSON("http://localhost/jqueryui/contacts.php
?jsoncallback =?", function(data) {
```

```
    //create arrays for data
  var names = [];
    var mails = [];
    //populate arrays
    for (var x = 0; x < data.contacts.length; x++) {
      names.push(data.contacts[x].name);
      mails.push(data.contacts[x].mail);
    }
    //add mail address to selected name
    function mailMatch(data, i, max) {
      return data + " (" + mails[i - 1] + ")"
    }
    //create config
    var autocompOpts = {
      data: names,
      width: 706,
      multiple: true,
      multipleSeparator: "; ",
      formatMatch: mailMatch
    };
    //turn specified element into auto-complete
    $("#address").autocomplete(autocompOpts);
  });
});
```

We're using a combination of both remote and local data in this example. First, we use jQuery's `getJSON` method to retrieve a JSON object outputted by a PHP file. We use a JSONP callback that processes the returned JSON following a successful request.

The JSON object will be structured in exactly the same way as the object we used in the AJAX date picker earlier in the book, and will be an array where each item in the array is an object containing `name` and `mail` keys where the contact's name is the value of `name` and their email address is the value of `mail`.

The data from the JSON object is used to populate two arrays. The first array will consist of each of the name values from each item, and the `mails` array will contain the mail addresses. Because every value in the database that is pulled has two properties, the items in each array will always be in sync, so `names[1]` will always match `mails[1]`.

We then add a custom function to format the data that is added to the `<textarea>`. The auto-complete will use the local `names` array as the data source. Therefore, each time a name is selected from the auto-complete suggestion menu, the matching email address is added.

After this, we create the configuration object used to make auto-complete do the things we want, such as adding support for multiple selections, and then initialize the widget as normal. We obviously need a PHP file as well to get the data out of the database in order to pass it back to our JSONP callback. Again, this is being provided in a supplementary manner:

```
<?php
  header('Content-type: application/json');
   //connection information
  $host = "localhost";
  $user = "root";
  $password = "your password here";
  $database = "autoComp";
  //make connection
  $server = mysql_connect($host, $user, $password);
  $connection = mysql_select_db($database, $server);
  //protect against sql injection
  mysql_real_escape_string($param, $server);
  //query the database
  $query = mysql_query("SELECT * FROM contacts");
   //start JSON object
   $contacts = "({ 'contacts':[";
  //loop through and return matching entries
  for ($x = 0; $x < mysql_num_rows($query); $x++) {
    $row = mysql_fetch_assoc($query);
      $contacts .= "{'name':'" . $row["name"] . "','mail':'" .
$row["mail"] . "'}";
  //add comma if not last row, closing brackets if is
  if ($x < mysql_num_rows($query) -1)
    $contacts .= ",";
  else
    $contacts .= "]})";
  }
  //return JSON with GET for JSONP callback
  $response = $_GET["jsoncallback"] . $contacts;
  echo $response;
  mysql_close($server);
?>
```

I understand that you may not have the setup to run this file, or have any interest in learning or using PHP, and that's fine. Partly, the reason for using dynamic data locally via JSON and a JSONP callback is that you can still use this example without having your own local web server. To this end, I have placed a copy of this file on my own web server. To use it, simply change the URL in the getJSON method to `http://www.danwellman.co.uk/jqueryui/contacts.php`.

# Summary

The auto-complete is a fantastic and very fresh addition to the jQuery UI library. It's intelligent, attractive, and intuitive to use. As we've seen with the other library components, it's also easy to use from a developer's perspective and flexible enough to be tailored to many individual situations.

Your visitors will love it because it makes arduous tasks, like the completion of forms, easier and quicker. It also improves the appearance and overall functionality of your site while lending an air of quiet professionalism.

We saw that this component brings to the table an impressive number of configurable properties that allows us to fine-tune the user-experience with high precision. We can use advanced formatting and matching properties to control the data that is outputted and how matching is performed, customize the appearance of the drop-down suggestion menu, and much more.

We also looked at the methods exposed by this component, including the usual `destroy` method for removing functionality. We can also programmatically flush the local cache and provide a new configuration object.

Unlike most of the other components we have seen so far, the auto-complete event model is method-based instead of property-based. However, it is equally as effective at allowing us to react to important events fired during an interaction.

# 8
# Drag and Drop

So far in this book, we've covered the complete range of fully released interface widgets, as well as one still (at the time of writing) in its beta phase. Over the next four chapters, we're going to shift our focus to the core interaction helpers. These components of the library differ from those that we've already looked at in that they are not physical objects that exist on the page.

Instead, they give an object a set of generic behaviors to suit common implementational requirements. You don't actually see them on the page, but the effects that they add to different elements, and how they cause them to behave, can easily be seen. These are low-level components as opposed to the high-level widgets. There are currently five different interaction helpers, each catering for a specific interaction.

They help the elements used on your pages to be more engaging and interactive for your visitors, which adds value to your site and can help make your web applications appear more professional. They also help to blur the distinction between the browser and the desktop as application platforms.

In this chapter, we'll be covering two very closely related components— draggables and droppables. The draggables API transforms any specified element into something that your visitors can pick up with the mouse pointer and drag around the page. Methods are exposed which allow you to restrict the draggables movement, make it return to its starting point after being dropped, and much more.

The droppables API allows you to define a region of the page, or a container of some kind, for people to drop the draggables on to in order to make something else happen. For example, to define a choice that is made, or add a product to a shopping basket. A rich set of events are fired by the droppable that lets us react to the most interesting moments of any drag interaction.

The full range of subjects we'll be covering in this chapter are:

- How to make elements draggable
- How to determine the new position of an element that has been dragged
- The properties that are available for configuring draggable objects
- How to make an element return to its starting point once the drag ends
- How to use event callbacks at different points in an interaction
- The role of a drag helper
- Containing draggables
- How to control draggability with the component's methods
- Turning an element into a drop target
- Defining accepted draggables
- Working with droppable class names
- Defining drop tolerance
- Reacting to interactions between draggables and droppables

# The deal with drag and droppables

We'll be devoting ourselves to these two components for the duration of this chapter because of how closely related they are. Dragging and dropping, as behaviors, go hand-in-hand with each other. Where one is found, the other is invariably close by. It's all very well dragging an element around a web page, but if there's nowhere for that element to be dragged to, or on top of, then the whole exercise is pointless.

You can use the draggable class completely independent of the droppable class as pure dragging, for the sake of dragging, can have its uses, such as with the dialog. However, you can't use the droppable class with the draggable. You don't need to make use of any of draggables methods of course, but using droppables without having anything to drop onto them is of no value.

These two components aren't designed to be used beyond simple drag and drop scenarios (which in themselves are complex pieces of web mechanics). If you have a more advanced requirement, like reordering list-based elements for example, you'll need to turn to a more specialized class, like the sortables component that we'll be looking at in the next chapter.

# Draggables

The draggables component is used to make any specified element, or collection of elements, draggable, so that they can be 'picked up' and moved around the page by a visitor. Draggability is a great effect, and is a feature that can be used in numerous ways to improve the interface of our web pages.

Using jQuery UI means that we don't have to worry about all the tricky differences between browsers that originally made draggable elements on web pages a nightmare to implement and maintain.

# A basic drag implementation

Let's look at the default implementation by first making a simple image draggable. We won't do any additional configuration. Therefore, all this code will allow you to do is 'pick up' the image with the mouse pointer and drag it around the viewport.

The element is made to appear draggable by having its left and top style properties manipulated in line with the mouse pointer. We don't need to worry about how this is achieved by the library, thanks to the object-oriented concept of encapsulation.

In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 1</title>
  </head>
  <body>
    <div id="drag"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
```

```
    $(function() {

      //make the specified element draggable
      $("#drag").draggable();
    });
  </script>
 </body>
</html>
```

Save this as `draggable1.html` in your `jqueryui` folder. As with the widget-based components of jQuery UI, the draggables component can be enabled using a single line of code. This invokes the draggables constructor method `draggable` and turns the specified element into a draggable object

We need the following files from the library to enable draggability:

- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.draggable.js`



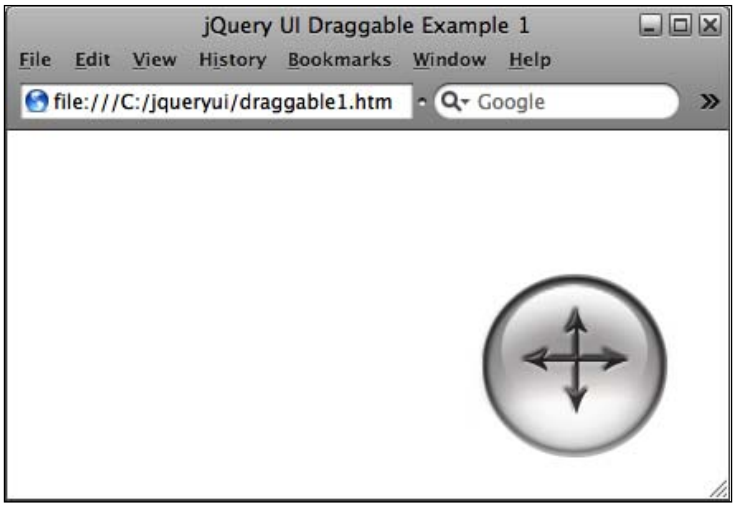
**Encapsulation**

Encapsulation is an object-oriented term that simply means we don't need to know how any particular module of code works internally. All we need to know is how to use the API that it exposes.

We're using a plain `<div>` with a background image specified in the CSS file we're linking to in the `<head>` of the page. Use the following stylesheet for the draggable element:

```
#drag {
  background:url(../img/draggable.png) no-repeat;
  width:114px;
  height:114px;
  cursor:move;
}
```

Save this as `draggable.css` in your `styles` folder. When you view the page in a browser, you'll see that the image can be moved around to your heart's content, as shown in the following screenshot:




# Configuring draggable properties

The draggables extension has a wide range of configurable properties, giving you a very fine degree of control over the behavior that it adds. The following table lists the properties that we can manipulate to configure and control our draggable elements:

| Property | Usage |
|---|---|
| `appendTo` | Specifies a container element for draggables with a helper attached |
| `axis` | Constrains draggables to one axis of motion and can take the strings `x` and `y` as values |
| `cancel` | Prevents certain elements from being dragged if they match the selector |
| `containment` | Prevents draggables from being dragged out of the bounds of its parent element |
| `cursor` | Specifies a CSS `cursor` to be used with the draggable |
| `cursorAt` | Specifies a default position at which the cursor appears relative to the draggable while it is being dragged |
| `delay` | Specifies a time in milliseconds for the start of the drag to be delayed |
| `distance` | Specifies the distance in pixels that the pointer should move with the mouse button held down on the draggable before drag begins |
| `grid` | Makes the draggable snap to an imaginary grid on the page |

| Property | Usage |
|---|---|
| `handle` | Defines a specific part of the draggable which is used to hold the pointer on in order to drag. |
| `helper` | Defines a pseudo-drag element which is dragged in place of the draggable |
| `opacity` | Sets the `opacity` of the helper element |
| `revert` | Makes the draggable return to its start position once dragging ends |
| `scroll` | Makes the draggables container automatically scroll |
| `scrollSensitivity` | Defines how close the draggable should get to the edge of the viewport before it begins to scroll |
| `scrollSpeed` | Sets the speed at which the viewport scrolls |
| `snap` | Causes drag objects to snap to the edges of specified elements |
| `snapMode` | Can be set to either `inside`, `outside`, or `both`, with `both` being the default |
| `snapTolerance` | The distance from snapping elements that draggables should reach before snapping occurs |
| `refreshPositions` | Calculates all draggable positions on every mouse move |
| `zIndex` | Sets the `z-index` of the helper element |

Let's put some of these properties to use. They can be configured in exactly the same way as the properties of the widgets that we looked at in previous chapters. This is done by creating a literal object with the chosen properties and their values configured, and passing this object into the draggable constructor method.

In the first example a moment ago, we used CSS to specify that the `move` cursor should be used when the pointer hovers over our draggable `<div>`. Let's change this and use the `cursor` property of the draggables component instead. Remove `cursor:move` from `draggable.css` and change `draggable1.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableNoCursor.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 2</title>
  </head>
  <body>
    <div id="drag"></div>
```

```
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
      var dragOpts = {
        cursor: "move"
      };

       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable2.html` and try it out in your browser. An important point to note about this property is that the `move` cursor we have specified is not applied until we actually start the drag. When using this property in place of simple CSS, we should perhaps provide some other visual cue that the element is draggable on mouse-over.

Let's look at a few more of draggable's many properties. Change `draggable2.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableNoCursor.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 3</title>
  </head>
  <body>
    <div id="drag"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
      var dragOpts = {
        cursor: "move",
        axis: "y",
        distance: "30",
        cursorAt: {
          top:0,
          left:0
          }
      };

       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

This can be saved as `draggable3.html`. The first new property that we've configured is the `axis` property, which has restricted the draggable to moving only up or down the page, but not side-to-side.

Next, we've specified `30` as the value of the `distance` property. This means that the cursor will have to travel 30 pixels across the draggable, with the mouse button held down, before the drag will begin.

The final property, `cursorAt`, is configured using an object whose properties can be `top`, `right`, `bottom`, or `left`. The values supplied to the properties we choose to use are the values relative to the draggable object that the cursor will assume when a drag occurs.

You'll notice in this example however, that the value for the `left` property seems to be ignored. The reason is that we have configured the `axis` property. When we begin the drag, the draggable will automatically move so that the cursor is at `0` pixels from the `top` of the element, but it will not move so that the cursor is `0` pixels from the `left` edge as we have specified. If we comment out the `axis` property, the cursor will then behave as expected.

Let's look at some more of draggable's properties in action. Change `draggable3.html` so that the configuration object appears as follows:

```
//define config object
var dragOpts = {
  delay: "500",
  grid: [100,100]
};
```

Save the file as `draggable4.html`. The `delay` property, which takes a value in milliseconds, configures the time that the mouse button must be held down with the cursor over the draggable object before the drag will begin.

The `grid` property, which is similar in usage to the `steps` property of the slider widget, is configured using an array of two values representing the number of pixels along each axis the drag element should jump when it is dragged. This property can be used safely in conjunction with the `axis` property.

# Resetting dragged elements

It is very easy to configure draggables to return to their original starting position on the page when the draggable has been dropped. Let's look at this behavior in action. Change `draggable4.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableNoCursor.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 5</title>
  </head>
  <body>
    <div id="drag"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
```

```
        //define config object
        var dragOpts = {
          revert: true
        };

        //make the specified element draggable
        $("#drag").draggable(dragOpts);
      });
    </script>
  </body>
</html>
```

Save this as `draggable5.html`. By supplying `true` as the value of the `revert` property, we've caused the draggable to return to its starting position at the end of any drag interaction. You'll notice, however, that the drag element doesn't just pop back to its starting position instantly. Rather, it's smoothly animated back with no additional configuration required.

# Drag handles

The `handle` property allows us to define a region of the draggable object which can be used to drag the object. All other parts of the draggable cannot be used to drag the object. A simple analogy is the dialog widget. You can drag the dialog around only if you click and hold on the title bar. The title bar is the drag handle.

In the following example, we'll add a simple drag handle to our draggable. In a new page in your text editor, add the following page:

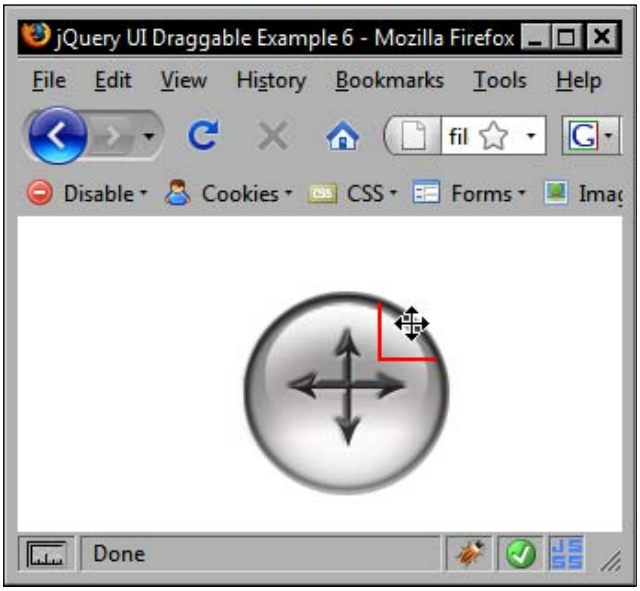
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
dragHandle.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 6</title>
  </head>
  <body>
    <div id="drag"><div id="handle"></div></div>
```

```
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
      var dragOpts = {
        handle:  "#handle"
      };

       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable6.html`. We've added another `<div>` element within our draggable in the mark-up, and used a jQuery selector as the value of the `handle` property to target the new element. The handle is styled with a few simple style rules. Create a new stylesheet and add to it the following code:

```
#drag {
  background:url(../img/draggable.png) no-repeat;
  width:114px; height:114px;
}
#handle {
  border-bottom:2px solid #ff0000;
  border-left:2px solid #ff0000;
  position:absolute;
  right:10px; top:10px;
  width:30px; height:30px;
  cursor:move;
}
```

Save this as `dragHandle.css` in the `styles` folder. When we run the page in a browser, we see that the original drag object is still draggable, but only when the handle is selected with the pointer as seen here:




# Helper elements

Several configuration properties are directly related to drag helpers or draggable objects used with helpers. A helper is a substitute element that is used to show where the object is on screen while the drag is in progress, instead of moving the actual draggable.

A helper can be used with a very simple object in place of the actual draggable. This prevents the client computer from needing to maintain the position of a memory heavy, complex object, which can consume a high number of CPU cycles. Once the drag has completed, the actual element can be moved to the new location.

Let's look at how they are used with the help of the following example. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableNoCursor.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 7</title>
  </head>
  <body>
    <div id="drag"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var dragOpts = {
          helper: "clone"
        };

        //make the specified element draggable
        $("#drag").draggable(dragOpts);
      });
    </script>
  </body>
</html>
```

The value `clone` for the `helper` property causes an exact copy of the original draggable to be created and used as the draggable. Therefore, the original object stays in its starting position at all times. This also causes the clone object to revert back to its starting position, an effect which cannot be changed, even by supplying `false` as the value of the `revert` property. The following screenshot shows the `clone` property in action:

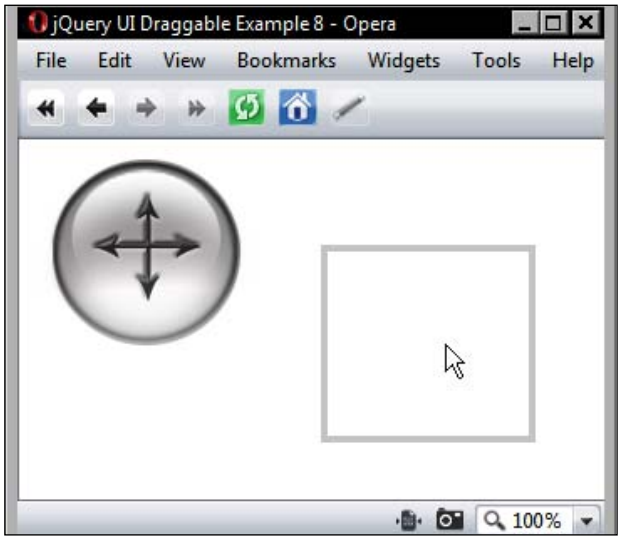


Save this change as `draggable7.html`. In addition to the `clone` string, and the default string value of `original`, we can also use a function as the value of this property. This allows us to specify our own custom element to use as the helper. Change `draggable7.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 8</title>
  </head>
  <body>
    <div id="drag"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery.ui-1.5b4"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
       //define config object
      var dragOpts = {
        helper:helperMaker
      };
      //define function that returns helper element
      function helperMaker() {
        return $("<div>").css({
          border: "4px solid #cccccc",
          opacity: "0.5",
          height: "110px",
          width: "120px"
        });
      }
       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable8.html`. Our `helperMaker()` function creates a new `<div>` element using standard jQuery functionality, and then sets some CSS properties on it to define its physical appearance. It then, importantly, returns the new element. When supplying a function as the value of the `helper` property, the function must return an element.

Now, when the drag begins, it is our custom helper that becomes the draggable. Because our custom element is much simpler than the original drag object, it can help improve the responsiveness and performance of an application it is used in.






We used the `css` jQuery method in this example during the creation of the custom helper. However, we can also use the `opacity` property of the draggable to set the opacity of helper elements as a cross-platform solution.

We used an older version of the library in the above example. This is due to a bug in 1.6rc2 version of the library. By the time you read the book, this bug will hopefully have been eradicated and the example will work with the latest stable release of the library.

# Constraining the drag

Another aspect of drag scenarios is that of containment. In our examples so far, the `<body>` of the page has been the container of the draggable. There are also properties that we can configure to specify how the draggable works with regard to another container element. We'll look at these in the following examples, starting with the `container` property which allows us to specify a container element for the draggable.

Add the following code to a new page in your text editor:

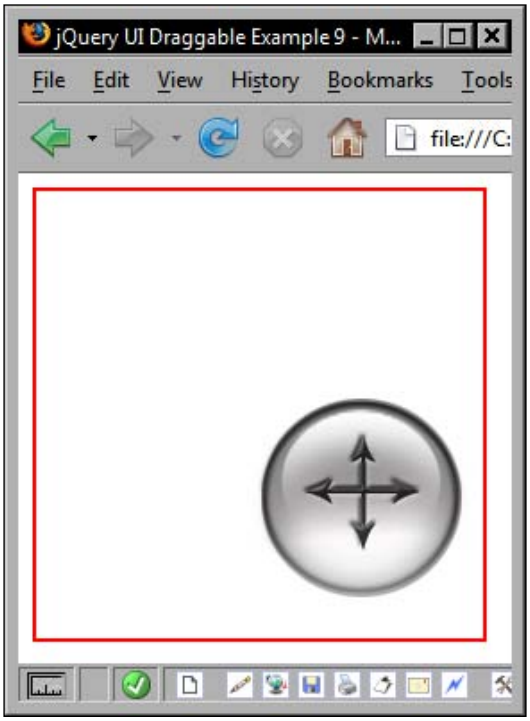
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableContainer.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 9</title>
  </head>
  <body>
    <div id="container"><div id="drag"></div></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
       var dragOpts = {
         containment: "parent"
       };

       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable9.html`. On the page, we've added a new `<div>` element as the parent of the existing draggable. In the code, we've used the value `parent` for the `containment` property.

The parent `<div>` has been given some basic styling to give it dimensions and so it can be seen. Add the following line of code to `draggable.css` and resave the file as `draggableContainer.css`:

```
#container {
  height:250px; width:250px;
  border:2px solid #ff0000;
}
```

When you run the page in your browser, you'll see that the draggable cannot exceed the boundary of its container:




There are three additional properties related to draggables within containers which are related to scrolling. However, you should note that these are only applicable when the document is the container.

The default value of the `scroll` property is `true`, but when we drag the `<div>` to the edge of the container, it does not scroll. You may have noticed in previous examples, where the draggable was not within a specified container, the viewport does automatically scroll.

You will also note that the previous example, at the time of writing, seems to have problems when run in Safari or Chrome. Using the `container` property constrains the draggable to the `y` axis in these browsers.

# Snapping

Draggable elements can be given an almost magnetic quality by configuring snapping. This feature causes dragged elements to snap to specified elements while they are being dragged. There are three properties related to snapping:

- `snap`
- `snapMode`
- `snapTolerance`

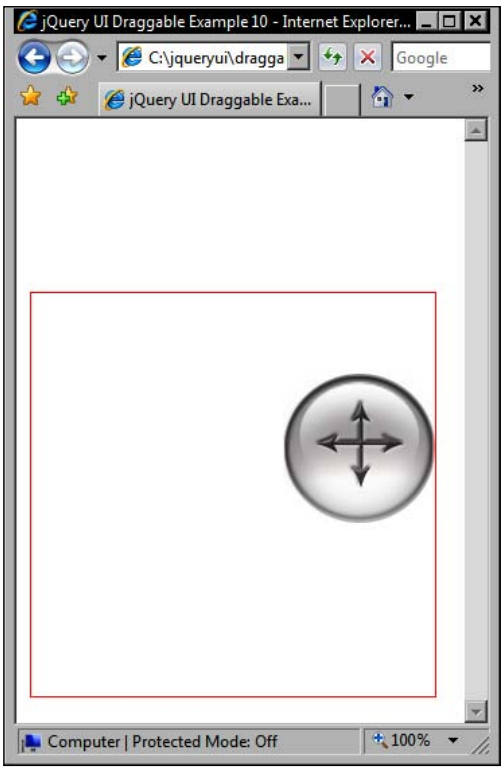
In the next example, we'll look at the effects that these properties have on the behavior of the draggable when they are configured. In a new file in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableSnap.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 10</title>
  </head>
  <body>
    <div id="drag"></div>
    <div id="snapper"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
      var dragOpts = {
        snap: "#snapper",
        snapMode: "inner",
```

```
            snapTolerance: 50
          };

          //make the specified element draggable
          $("#drag").draggable(dragOpts);
        });
    </script>
  </body>
</html>
```

Save this as `draggable10.html`. We've supplied the selector `#snapper` as the value of the `snap` property. Therefore, our draggable will snap to this element on the page while the object is being dragged. We also set the `snapMode` property to `inner` (the other possible values are `outer` and `both`), so snapping will occur on the inside edges of our `snapper` element.

Finally, we've set the `snapTolerance` to `50`, which is the maximum distance (in pixels) the draggable will need to get to the snapper element before snapping will occur. Now, when you drag the image within `50` pixels of an edge of the `snapper` element, the draggable will automatically align itself to that edge, as shown in the following screenshot:



# Draggable event callbacks

In addition to the properties that we have already looked at, there are three more properties that can be used as callback functions to execute code after specific custom events, defined by the draggables component, occur. These events are listed below:

| Property | Triggered |
|---|---|
| `drag` | When the mouse is moved while dragging |
| `start` | When dragging starts |
| `stop` | When dragging stops |

When defining callback functions to make use of these events, the functions will always automatically receive two arguments. The original event object as the first argument and a second object containing the following properties:

| Property | Usage |
|---|---|
| `options` | The configuration object used with the draggable |
| `helper` | A jQuery object representing the helper element |
| `position` | A nested object with properties top and left of the helper element relative to the original drag element |
| `absolutePosition` | A nested object with properties top and left of the helper element relative to the page |

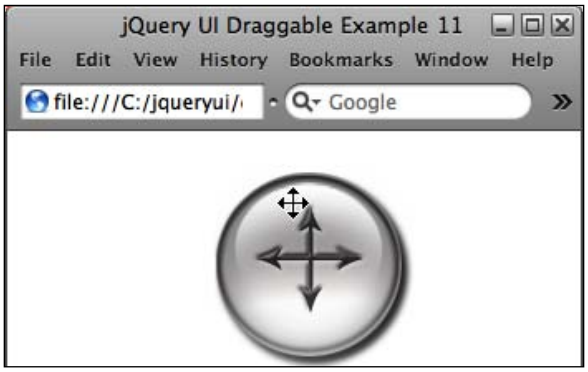
Using the callbacks, and the two objects they are passed as arguments, is extremely easy. We can look at a brief example to highlight their usage. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 11</title>
  </head>
  <body>
```

```
    <div id="drag"></div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //define config object
      var dragOpts = {
        start: setShadow,
        stop: unsetShadow
      };

      //swap image to add drop shadow
      function setShadow() {
        $("#drag").css({ background:"url(img/draggable_on.png)",
width:"120px", height:"121px" });
      }

      //swap image back to original
      function unsetShadow() {
        $("#drag").css({ background:"url(img/draggable.png)",
width:"114px", height:"114px" });
      }

       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable11.html`. In this example, our configuration object contains just two properties—the `start` and `stop` callbacks. We set these values to our callback function names.

All the functions do in this example are simple image swaps. When the `start` callback is invoked, the background image of the draggable is switched for one containing a drop shadow. When the `stop` callback is invoked, the image is swapped back to the original image with no shadow. The following screenshot shows the shadow:




Using the callbacks in this way is just one example of how the usability of the drag object is improved with a visual cue that indicates the object is currently in a draggable state.

Let's move on to a slightly more complex example where we can make use of the second object passed to our callbacks. Create a new page in your text editor and add to it the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggableIndented.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 12</title>
  </head>
  <body>
    <div id="container">
      <div id="drag"></div>
    </div>
    <div id="results"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
       //define config object
      var dragOpts = {
        helper: "clone",
        stop: getNewPos
       };
       //get the new position
       function getNewPos(e, ui) {
         var relativeP = $("<p>").attr("id", "test").text("The
helper was moved to " + ui.position.top + "px from the top, and " +
ui.position.left + "px to the left of the original object.");
         var absoluteP = $("<p>").attr("id", "test").text("The helper
was moved to " + ui.absolutePosition.top + "px from the top, and " +
ui.absolutePosition.left + "px to the left relative to the page.");
         $("#results").empty().append(relativeP).append(absoluteP);
        }
       //make the specified element draggable
       $("#drag").draggable(dragOpts);
     });
   </script>
  </body>
</html>
```

Save this as `draggable12.html`. We're now using a helper with our draggable, which is necessary in this implementation, as I'll explain in a moment. We've defined the `getNewPos` callback function as the value of the `stop` property, so it will be executed each time a drag interaction stops.

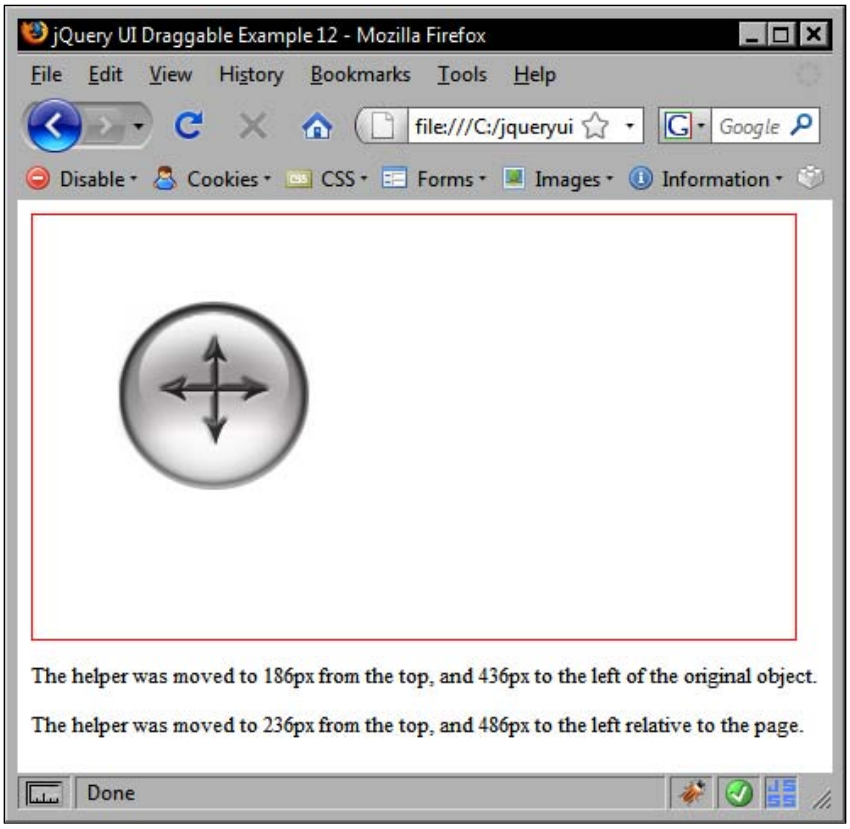
Our callback function receives the object as `e` for the event object (which we don't need but must specify in order to get to the second object), and `ui` for the jQuery UI object containing useful information about the draggable and its helper.

All our callback function does is create two new paragraphs, concatenating in the values found in the object passed to the function as the second argument- `ui.position.top`, `ui.position.left`, `ui.absolutePosition.top`, and `ui.absolutePosition.left`. It then inserts the new `<p>` elements into the results `<div>`. For reference, these positional properties are only available when using a helper object. A brief stylesheet has been also been used, which should be as follows:

```
#container {
  width:450px; height:250px;
  border:1px solid #ff0000;
```

```
}
#drag { background:url(../img/draggable.png) no-repeat;
  width:114px; height:114px;
  margin-left:50px; margin-top:50px;
  cursor:move;
}
p { font-size:80%; }
```

This should be saved as `draggableIndented.css` in the `styles` folder. It is necessary to give the drag element margins in this example, so the differences between the `position` and `absolutePosition` properties are shown. If we didn't do this, the new `<p>` elements would both contain the same text. Here's how it should look after dragging to the bounds of the container:



# Using draggable's methods

Three methods (not including the constructor) are defined for draggables:

- `destroy`
- `enable`
- `disable`

These methods are used in the same way as the methods for the widgets that we've already used, but we'll look at a brief example for the sake of completeness. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
draggable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Example 13</title>
  </head>
  <body>

    <button id="disable">Disable</button>
    <button id="enable">Enable</button>
    <button id="destroy">Destroy</button>
    <div id="drag"></div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

      //make the specified element draggable
      $("#drag").draggable();

      //define function to toggle draggability
      function toggle(action) {
        (action == "destroy") ? $("#drag").draggable(action)
.removeClass("drag") : $("#drag").draggable(action);
      }
```

```
        //define click handler for buttons
        $("button").click(function() {
          toggle($(this).attr("id"));
        });
      });
    </script>
  </body>
</html>
```

Save this as `draggable13.html`. The page contains three buttons, in addition to the draggable, which will be used to drive the method functionality in this example.

The script is also very straightforward. Whenever a button is clicked, we simply get the `id` of the button and call the `toggle()` function, passing in the `id` we just obtained from the button as an argument.

The `toggle()` function then calls the method specifying the string it received as an argument. This way, it doesn't matter which button gets clicked, the appropriate method will be called.

There is also an additional layer of checking that is done with the JavaScript ternary. Unfortunately, the functionality of the `destroy` method is, for all intents and purposes, synonymous with that of the `disable` method.

If the **Destroy** button is clicked, the function will remove the class name `drag` from the draggable so the object loses all of its style properties and vanishes from the page (although it still exists in the DOM).

There won't be a '*fun with*' section that focuses solely on the use of the draggable component alone. Because draggable and droppable work so well together, we'll have a combined '*fun with*' section involving both components at the end of the chapter. Let's continue by moving on to the droppable component.

# Droppables

Making elements draggable adds a level of interactivity and versatility to your web pages unmatched by almost any other DHTML technique. Being able to define valid targets for draggable objects, by using the droppables component, throws logic into the mix as well. For a draggable to have some semblance of practicality, it should have somewhere that it can be dragged *to* which causes something else to happen.

In a nutshell, this is what the droppables component of jQuery UI achieves. It gives you a place for draggable elements to be dropped. A region of the page is defined as a droppable, and when a draggable is dropped onto that region, something else is triggered. You can react to drops on a valid target very easily using the extensive event model.

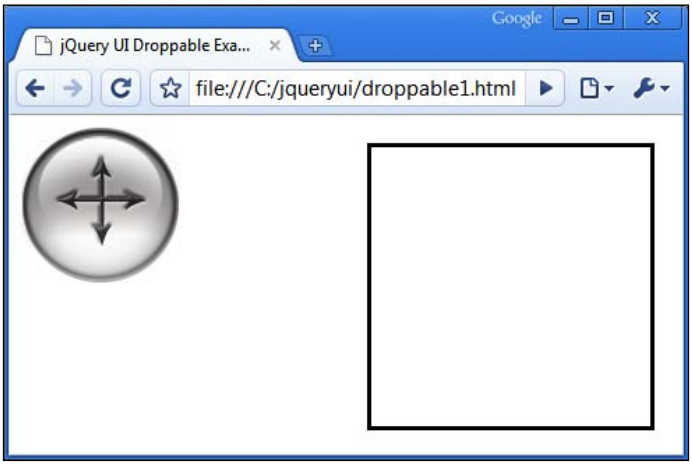
Let's start with the default droppable implementation. In a new file in your text editor, add the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Droppable Example 1</title>
  </head>
  <body>
    <div id="drag"></div>
    <div id="target"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {

       //make the specified element draggable
       $("#drag").draggable();

      //make the target droppable
      $("#target").droppable();
     });
   </script>
  </body>
</html>
```

Save this as `droppable1.html`. The extremely basic stylesheet that is linked to in this example is simply an updated version of `draggable.css` and appears as follows:

```
#drag {
  background:url(../img/draggable.png) no-repeat;
  width:114px; height:114px;
  cursor:move; margin-bottom:5px;
  z-index:2;
}
#target {
  width:200px; height:200px;
  border:3px solid #000;
  position:absolute;
  right:20px; top:20px;
  z-index:1;
}
```

Save this as `droppable.css` in the `styles` folder. When run in a browser, the page should look like the following screenshot:




The default droppable implementation literally does nothing. In this example, the droppable is created (we can see this with the class name `target ui-droppable` which is added to the specified element at run time), but other than this, nothing happens at all, even when a draggable is dropped onto it.

When I say that nothing happens, I mean that we haven't added any code which will allow us to see things happen. Events are still firing throughout the interaction on both the draggable and the droppable. A little later in the chapter we'll look at these events in more detail.

The files we used for this basic droppable implementation are:

- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.draggable.js`
- `ui.droppable.js`

As you can see, the droppables component is an extension of draggables, rather than a completely independent component. Therefore, it requires the `ui.draggable.js` file in addition to its own source file. The reason our droppable does nothing is that we haven't configured it, so let's get on and do that next.

# Configuring droppables

The droppable class is considerably smaller than the draggables class and there are less configurable properties for us to play with. The following table lists those properties available to us:

| Property | Usage |
|---|---|
| `accept` | Sets the element(s) that the droppable will accept |
| `activeClass` | The class that is applied to the droppable while an accepted draggable is being dragged |
| `greedy` | Used to stop drop events from bubbling when a draggable is dropped onto nested droppables |
| `hoverClass` | The class that is applied to the droppable while an accepted draggable is hovering over the droppable |
| `tolerance` | Sets the mode that triggers an accepted draggable being considered over a droppable. |

In order to get a visible result from configuring the droppable, we're going to use a couple of these properties together in the following example that will highlight the droppable that accepts the draggable. Change `droppable1.html` so that it appears as follows (new code is shown in bold):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
```

```
    <title>jQuery UI Droppable Example 2</title>
  </head>
  <body>
    <div class="drag" id="drop1"></div>
    <div class="drag" id="drop2"></div>
    <div id="target"></div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
       //make the specified element draggable
       $(".drag").draggable();
       //define config object
       var dropOpts = {
         accept: "#drop1",
         activeClass: "activated"
       };
       //make the target droppable
       $("#target").droppable(dropOpts);
     });
   </script>
  </body>
</html>
```

Save this as `droppable2.html`. The `accept` property takes a string that can be used as a jQuery selector. In this case, we've specified that only the draggable with an `id` of `drag1` should be accepted.

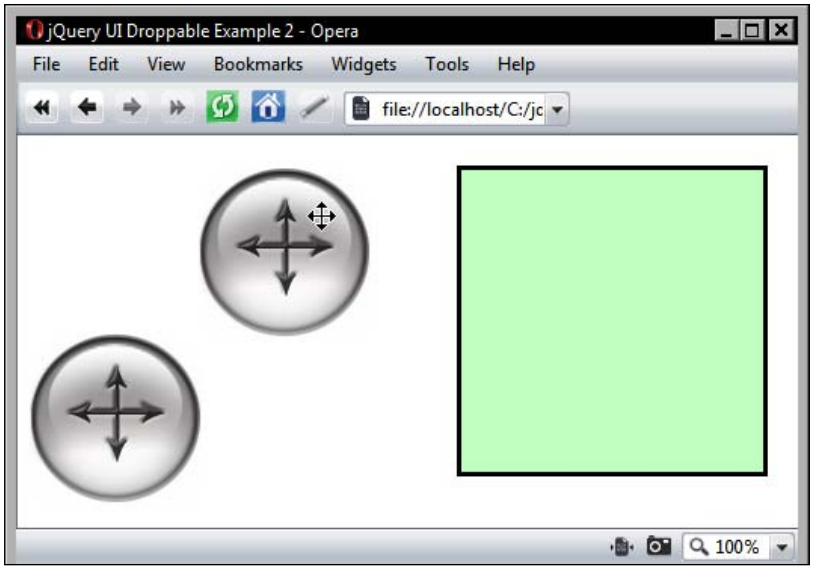
We've also specified the class name `activated` as the value of the `activeClass` property. This class name will be applied to the droppable when the draggable starts to be dragged. The `hoverClass` property can be used in exactly the same way to add styles when a draggable is over a droppable. The style rules that our `activated` class will pick up can be added to `droppables.css`:

```
.activated {
  border:3px solid #339900;
  background-color:#ccffcc;
}
```

We should also add a class selector to the rule that gives the drag elements the background image and size styles. Change the first line of code to this:

```
#drag, .drag {
```

When we view this page in a browser, we find that as we move the top draggable, which has been accepted, the droppable picks up the `activated` class and turns green. However, when the second draggable is moved, the dropper does not respond. The following screenshot shows how the page should look:




In addition to a string value, the `accept` property can also take the name of a function as its value. This function will be executed one time for every draggable that is on the page when the page load. It must return either `true`, to indicate that the draggable is accepted, or `false` to indicate that it's not. To see the function value of the `accept` property in action, change `droppable2.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Droppable Example 3</title>
  </head>
  <body>
```

```
    <div class="drag" id="drop1"></div>
    <div class="drag" id="drop2"></div>
    <div id="target"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //make the specified element draggable
        $(".drag").draggable();

       //define config object
       var dropOpts = {
         accept: dragEnrol,
         activeClass: "activated"
       };

       //set drop1 as acceptable
       function dragEnrol(el) {
         return (el.attr("id") == "drop1") ? true : false;
       }

       //make the target droppable
       $("#target").droppable(dropOpts);

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

Save this as `droppable3.html`. On the surface, the page works exactly the same as it did in the previous example. But this time, acceptability is being determined by the JavaScript ternary statement within the `dragEnrol` function, instead of a simple selector.

Note that the function is automatically passed an object containing useful data about the draggable element as an argument. This makes it is easy to obtain information about the draggable, like its `id` in this example. This callback can be extremely useful when advanced filtering, beyond a simple selector, is required.

# Tolerance

Drop tolerance refers to the way a droppable detects whether a draggable is over it or not. The default value is `intersect`. The following table lists the modes that this property may be configured with:

| Mode | Implementation |
|---|---|
| `fit` | The draggable must be completely within the boundary of the droppable for it to be considered over it |
| `intersect` | At least 25% of the draggable must be within the boundary of the droppable before it is considered over it |
| `pointer` | The mouse pointer must touch the droppable boundary before the draggable is considered over the droppable |
| `touch` | The draggable is over the droppable as soon as an edge of the draggable touches an edge of the droppable |

So far, all of our droppable examples have used `intersect`, the default value of the `tolerance` property. Let's see what difference the other values for this property make to an implementation of the component. In a new page in your text editor, add the following code:

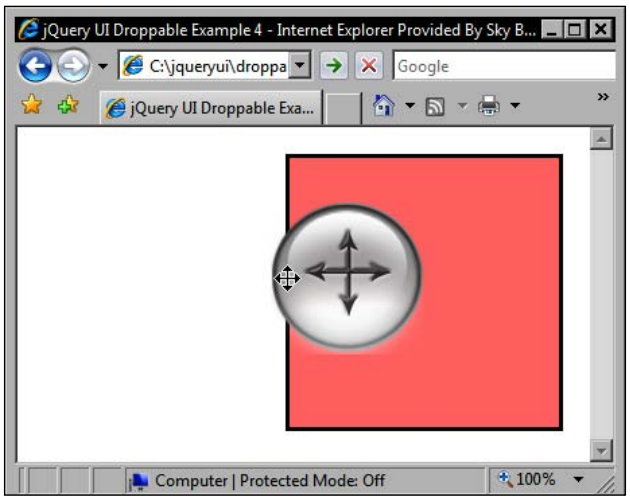
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Droppable Example 4</title>
  </head>
  <body>
    <div id="drag"></div>
    <div id="target"></div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
```

```
        //make the specified element draggable
        $("#drag").draggable();

        //define config object
       var dropOpts = {
         accept: "#drag",
         hoverClass: "over",
         tolerance: "pointer"
       };

       //make the target droppable
       $("#target").droppable(dropOpts);

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

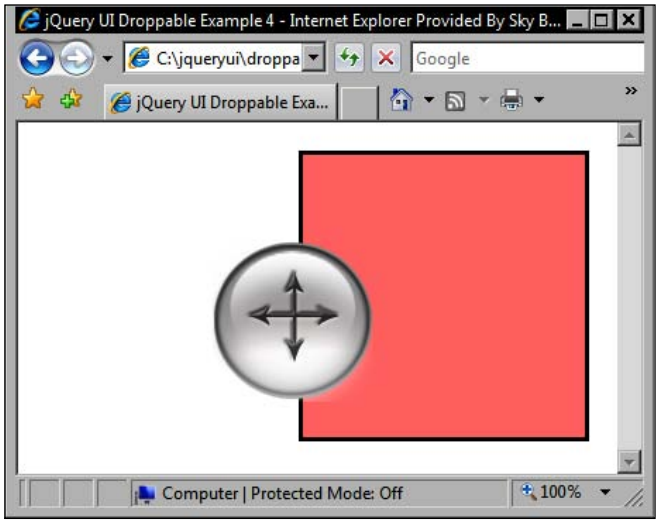
Save this as `droppable4.html`. We've specified the acceptable draggable for our droppable as any element with an `id` of `drag`. This property must be configured for the `hoverClass` property to have any effect. The `hoverClass` is necessary in this example so we can easily see when the draggable is considered to be over the droppable. It will be at this point that the styles specified in the `over` class are picked up.

The part of the draggable that is over the droppable is irrespective in this example. It is the mouse pointer that must cross the boundary of the droppable while a drag is in progress.

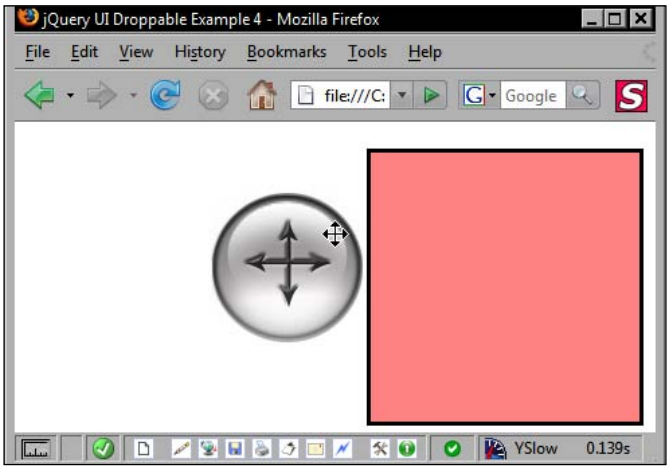
Note that the draggable must be active for our class to be applied. Now, with `pointer` specified as the value of the `tolerance` property, the mouse pointer, not just part of the draggable, must cross the droppable before our `over` class is applied:



If you comment out the `tolerance` property and run the example, you'll see that with the default value, at least a quarter of the area of the draggable must be within the boundary of the droppable for our `over` class to be applied:




For good measure, the following screenshot shows how the `touch` mode works. Here, the draggable need only to touch the edge of the droppable before triggering our `over` class:






You should note that neither Safari nor Chrome currently implements the `hoverClass` property correctly so these style changes will not be visible in these browsers.

# Droppable event callbacks

The properties that we've looked at so far configure various operational features of the droppable. In addition to these, there are almost as many callback properties so that we can define functions which react to different things occurring to the droppable and its accepted draggables. These properties are listed below:

| Callback Property | Invoked |
|---|---|
| `activate` | When an accepted draggable begins dragging |
| `deactivate` | When an accepted draggable stops being dragged |
| `drop` | When an accepted draggable is dropped onto a droppable |
| `out` | When an accepted draggable is moved out of the bounds (including the tolerance) of the droppable |
| `over` | When an accepted draggable is moved within the bounds (including the tolerance) of the droppable |

Let's put together a basic example that makes use of these callback properties. We'll add a status bar to our droppable that reports the status of different interactions between the draggable and the droppable. In a new file in your text editor, create the following page:

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppableCallbacks.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Droppable Example 5</title>
  </head>
  <body>
    <div id="drag"></div>
    <div id="target"></div>
    <div id="status"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
```

```
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //make the specified element draggable
        $("#drag").draggable();

        //define array of messages
        var dropOpts = {
          accept: "#drag",
          activate: eventCallback,
          deactivate: eventCallback,
          drop: eventCallback,
           out: eventCallback,
           over: eventCallback
        };

        //define object of status messages
        var eventMessages = {
          dropactivate: "A draggable is active",
          dropdeactivate: "A draggable is no longer active",
          drop: "An accepted draggable was dropped on the droppable",
          dropout: "An accepted draggable has been moved out of the
droppable",
          dropover: "An accepted draggable is over the droppable"
        };

        //determine event and write status message
        function eventCallback(e) {
          var message = $("<p>").attr("id", "message").
text(eventMessages[e.type]);
          $("#status").empty().append(message)
        }

        //make the target droppable
        $("#target").droppable(dropOpts);
      });
    </script>
  </body>
</html>
```

Save this file as `droppable5.html`. The body of the page contains our new status bar, which in this case is a simple `<div>` element. Our configuration now has all of the callback properties defined, and for efficiency, they all point to the same function. Like the class name properties, the callbacks must be configured along with an `accept` value.

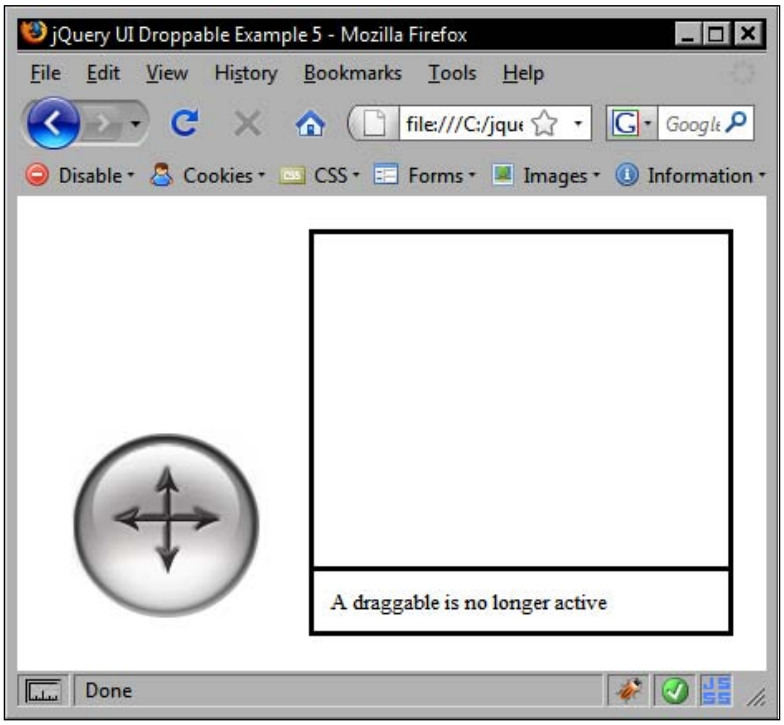
Following the configuration object, we define an object literal in which the name of each property is set to one of the event types that may be triggered. The value of each proptery is the message that we want to display for any given event.

Finally, we define our callback function. Like other components, the callback functions used in the droppables component are automatically passed two objects, the `event` object and an object representing the draggable. We use the `type` property of the event object to retrieve the appropriate message from the object. This is the same way we would access an associative array, and then use standard jQuery element creation and manipulation methods to add the message to the status bar.

We also use a new stylesheet for this example. Create a new stylesheet in your text editor and add the following selectors and rules:

```
#drag {
  background:url(../img/draggable.png) no-repeat;
  width:114px; height:114px;
  cursor:move;
  margin-bottom:5px;
  z-index:2;
}
#target {
  width:250px; height:200px;
  border:3px solid #000;
  position:absolute;
  right:20px; top:20px;
  z-index:1;
}
#status {
  width:230px;
  border:3px solid #000;
  position:absolute;
  top:223px; right:20px;
  color:#000;
  padding:10px;
}
#message {
  margin:0px;
  font-size:80%;
}
```

Here's how the status bar should look like following an interaction:




After playing around with the page for some time, we see that one of our messages does not appear to be working. When the draggable is dropped onto the droppable, our `drop` message does not appear.

Actually, the message does appear, but because the `deactivate` event is fired immediately after the `drop` event, the `drop` message is overwritten right away. There are a number of ways we could work around this, the simplest would be not to define the `deactivate` property.

# Greed

The final property that we are going to look at in connection with the droppable component is the `greedy` property. This property can be useful in situations where there is a droppable nested within another droppable. If we don't use this property, both droppables will fire events during certain interactions.

This is a situation faced at one point or another by most developers when working with traditional event models. It is a result of the way browsers propagate events (either event-bubbling or event-capturing, depending on the browser). The `greedy` property is an easy way to avoid event-bubbling problems in an efficient and cross-browser manner. Let's take a closer look at this property with an example. Create a new page in your text editor and add the following code to it:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
droppableNesting.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Droppable Example 6</title>
  </head>
  <body>
    <div id="drag"></div>
    <div class="target" id="outer">
      <div class="target" id="inner"></div>
    </div>
    <div class="status"></div>

<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //set opacity of targets
        $(".target").css({ opacity:"0.5" });

        //make the specified element draggable
        $("#drag").draggable();

        //define config object
        var dropOpts = {
         accept: "#drag",
         drop: dropCallback,
         greedy: true
```

```
        };
        //determine event and write status message
        function dropCallback(e) {
          var message = $("<p>").attr("id", "message").text("The
firing droppable was " + e.target.id);
          $("#status").append(message);
        }
        //make the target droppable
        $(".target").droppable(dropOpts);
      });
    </script>
  </body>
</html>
```

Save this example as `droppable6.html`. In this example, we have a smaller droppable nested in the center of a larger droppable. Their opacity is set using the standard jQuery library's `css()` method. In this example, this is neccesary because if we alter the `z-index` of the elements, so that the draggable appears above the nested droppables, the target element is not reported correctly.

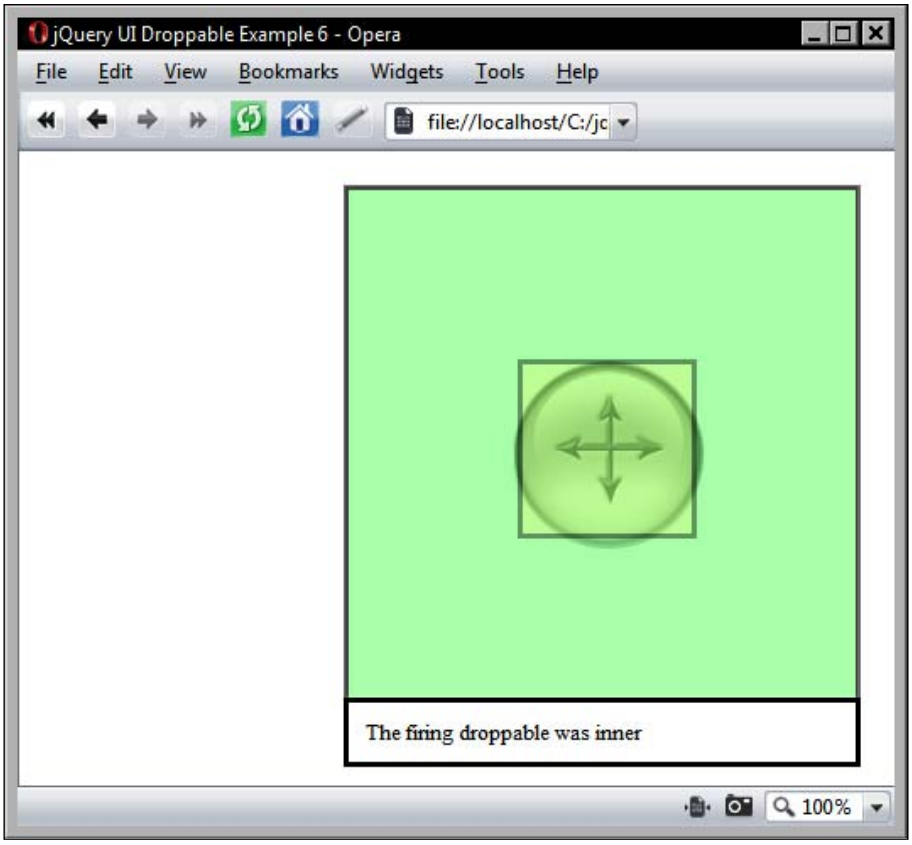
Our configuration object sets the `accept` and `drop` properties, in addition to the `greedy` property, which makes the droppables keep all of the event activity for themselves. Our callback function is then used to add a simple message to the status bar notifying us which droppable was the `target` of the drop.

The CSS for this example is simple and builds on the CSS of previous examples:

```
#drag {
  background:url(../img/draggable.png) no-repeat;
  width:114px; height:114px;
  cursor:move;
  margin-bottom:5px;
}
#outer {
  width:300px; height:300px;
  border:3px solid #000;
  position:absolute;
  right:20px; top:20px;
  background-color:#99FF99;
}
#inner {
  width:100px; height:100px;
  border:3px solid #000;
  position:relative;
```

```
    top:100px; left:100px;
    background-color:#FFFF99;
  }
  #status {
    width:280px;
    border:3px solid #000;
    position:absolute;
    top:323px; right:20px;
    color:#000;
    padding:10px;
  }
  #message {
    margin:0px;
    font-size:80%;
  }
```

Save this as `droppableNesting.css` in the `styles` folder. If you run the page, and drop the draggable onto one of the droppables, you should see something like this:



The net effect of setting the `greedy` property to `true` is that the inner droppable prevents the event from escaping into the outer droppable and firing again. If you comment out the `greedy` property, and drop the draggable onto the center droppable, the status message will be inserted twice, once by the inner droppable and once by the outer droppable.

# Droppable methods

Like the draggable component, droppable has only a few simple methods that we can make use of. This is another component that is primarily property-driven. The methods we have available are the same ones exposed by draggable:

- `destroy`
- `enable`
- `disable`

They function, and are used in exactly the same way as draggable. We can temporarily disable the droppable using the `disable` method, re-enable the droppable with `enable`, and permanently remove (at least for the duration of the session) functionality with `destroy`.

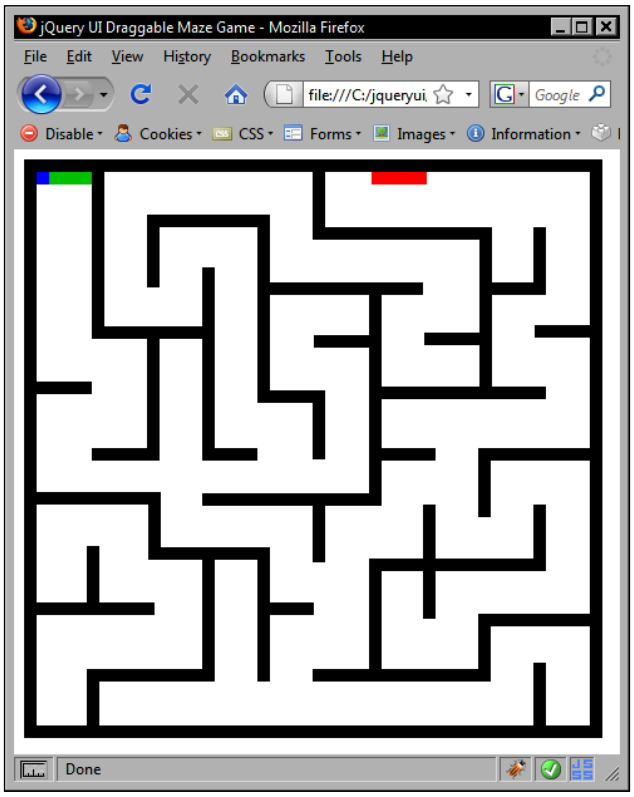
# Fun with droppables

We've now reached the point where we can have a little fun by putting what we've learned about these two components into a fully working example.

In our final drag and drop example, we're going to combine both of these components to create a simple maze game. It will be somewhat limited however. We're not going to have randomly generated maps and we won't be including AI enemies (the code payload would skyrocket were these to be a consideration).

The game will consist of a draggable marker which will need to be navigated through a simple maze to a specified droppable at the other end of the maze. We can make things a little more challenging so that if any of the maze walls are touched by the marker it will return to the starting position.

The following screenshot shows what we're going to build:




Let's start with the mark-up. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/dragMaze.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Draggable Maze Game</title>
  </head>
  <body>
    <div id="maze">
      <div id="drag"></div>
      <div id="start"></div>
      <div id="end"></div>
```

```
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.draggable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.droppable.js"></script>
    <script type="text/javascript">

    </script>
  </body>
</html>
```

On the page we have our outer container, which we've given an `id` of `maze`. We have `<div>` elements for the starting and ending positions as well as for the drag marker. Our map will need walls. Rather than hand-coding the 46 required walls for the map pattern that we're going to use, I thought we could use jQuery to do this for us instead.

We left an empty `<script>` element at the bottom of our page. Let's fill that up next with the following code:

```
//function to execute when doc ready
$(function() {

  //add map walls
  for (var x = 1; x < 47; x++) {
    $("<div>").attr({
      id: "a" + x,
      class: "wall"
    }).appendTo("#maze");
  }

  //define drag config object
  var dragOpts = {
    containment: "#maze"
  };

  //make the specified element draggable
  $("#drag").draggable(dragOpts);

  //define drop config object
  var dropOpts = {
    accept: "#drag",
    tolerance: "touch",
    over: function(e, ui) {
```

```
        //remove drag object from page
        $("#drag").draggable("destroy").remove();

        //create new draggable at start
        $("<div>").attr("id", "drag").css({ left:0, top:0
}).appendTo("#maze");

        //make the new element draggable
        $("#drag").draggable(dragOpts);
      }
    };

    //define end config object
    var endOpts = {
      accept: "#drag",
      over: function(e, ui) {

        //remove drag object from page
        $("#drag").draggable("destroy").remove();

        //congratulations
        alert("Wooot, you did it!");
      }
    };

    //make specified elements droppable
    $(".wall").droppable(dropOpts);

    //make end droppable
    $("#end").droppable(endOpts);

});
```

Let's review what the new code does. First, we use a simple `for` loop to add the walls to our maze. We use the plain-vanilla `for` loop in conjunction with standard jQuery to create 46 `<div>` elements and add `id` and `class` attributes to each one before appending them to the `maze` container.

We then define a simple configuration object for the draggable element. The only property we need to configure is the `container` property which constrains the draggable marker element within the maze. We can then go ahead and create the draggable behavior with the `draggable` constructor method.

Next, we can define the configuration object for the walls. Each wall is treated as a droppable that can accept the draggable marker element. We specify `touch` as the value of the `tolerance` property and add a callback function to the `over` property. Therefore, whenever the draggable touches a wall, the function will be executed.

All we do in this function is destroy the current draggable and remove it from the page. We then create a new draggable back at the starting position and make it draggable once more. There is no `cancelDrag` method which causes the draggable to act as if it had been dropped, but we can easily replicate this behavior ourselves.

We then add another droppable configuration object which configures the ending point of the maze. All we configure for this droppable is the element it accepts, which again is the draggable marker, and specify a function to execute when the draggable is over the end droppable. In this function, we remove the draggable again and present the user with an alert.

Finally, we make the walls and the end target droppables. So far, this is probably the simplest JavaScript game ever written, but we also need to add some CSS for the maze, the draggable, and the starting and ending points.

We also need to style up the walls of the maze, but we can't use any simple JavaScript pattern for this. Unfortunately, we have to hard-code them. In another new file in your text editor, add the following selectors and rules:

```
#maze {
  width:441px; height:441px; border:10px solid #000000;
  background-color:#ffffff; position:relative;
}
#drag {
  width:10px; height:10px;
  background-color:#0000FF; z-index:1;
}
#start {
  width:44px; height:10px; background-color:#00CC00;
  position:absolute; top:0; left:0; z-index:0;
}
#end {
  width:44px; height:10px; background-color:#FF0000;
  position:absolute; top:0; right:130px;
}
.wall { background-color:#000000; position:absolute; }
#a1 { width:10px; height:133px; left:44px; top:0; }
#a2 { width:44px; height:10px; left:0; top:167px; }
#a3 { width:44px; height:10px; left:44px; top:220px; }
#a4 { width:89px; height:10px; left:0; bottom:176px; }
#a5 { width:94px; height:10px; left:0; bottom:88px; }
#a6 { width:10px; height:41px; left:40px; bottom:0; }
#a7 { width:10px; height:48px; left:88px; top:44px; }
#a8 { width:78px; height:10px; left:54px; top:123px; }
#a9 { width:10px; height:97px; left:88px; top:133px }
```

```
#a10 { width:10px; height:45px; left:40px; bottom:98px; }
#a11 { width:88px; height:10px; left:89px; bottom:132px; }
#a12 { width:10px; height:97px; left:132px; bottom:35px; }
#a13 { width:10px; height:44px; left:89px; bottom:142px; }
#a14 { width:92px; height:10px; left:40px; bottom:35px; }
#a15 { width:89px; height:10px; left:88px; top:34px; }
#a16 { width:10px; height:145px; left:132px; top:76px; }
#a17 { width:44px; height:10px; left:132px; top:220px; }
#a18 { width:133px; height:10px; left:132px; bottom:175px; }
#a19 { width:10px; height:107px; left:176px; bottom:35px; }
#a20 { width:10px; height:150px; left:176px; top:34px; }
#a21 { width:35px; height:10px; left:186px; top:174px }
#a22 { width:35px; height:10px; left:186px; bottom:88px; }
#a23 { width:122px; height:10px; left:186px; top:88px; }
#a24 { width:10px; height:44px; left:220px; top:0px; }
#a25 { width:10px; height:55px; left:220px; top:174px; }
#a26 { width:10px; height:45px; left:220px; bottom:130px; }
#a27 { width:133px; height:10px; right:88px; top:44px; }
#a28 { width:10px; height:168px; right:166px; top:98px; }
#a29 { width:44px; height:10px; right:176px; top:130px; }
#a30 { width:10px; height:98px; right:166px; bottom:35px; }
#a31 { width:133px; height:10px; right:88px; bottom:35px; }
#a32 { width:10px; height:133px; right:78px; top:44px; }
#a33 { width:44px; height:10px; right:88px; top:128px; }
#a34 { width:131px; height:10px; right:35px; top:171px; }
#a35 { width:43px; height:10px; right:123px; top:220px; }
#a36 { width:10px; height:91px; right:123px; bottom:85px; }
#a37 { width:131px; height:10px; right:35px; bottom:123px; }
#a38 { width:10px; height:55px; right:79px; top:220px; }
#a39 { width:44px; height:10px; right:0; top:122px; }
#a40 { width:10px; height:54px; right:79px; bottom:35px; }
#a41 { width:79px; height:10px; right:0; bottom:79px; }
#a42 { width:10px; height:45px; right:35px; top:44px; }
#a43 { width:43px; height:10px; right:35px; top:88px; }
#a44 { width:79px; height:10px; right:0; top:220px; }
#a45 { width:10px; height:44px; right:35px; bottom:132px; }
#a46 { width:10px; height:50px; right:35px; bottom:0; }
```

Save this file as `dragMaze.css` in the `styles` folder. These two new files now form our simple game. It's limited, but you can see how well the drag and drop components work in this particular scenario.

We can now attempt to navigate the marker from the starting point to the finish by dragging it through the maze walls. If any wall is touched, the marker will return to the starting point. We could make it harder (by adding additional obstacles to navigate, etc), but for the purpose of having fun with jQuery UI draggables and droppables, our work here is complete.

# Summary

We looked at two very useful library components in this chapter—the draggable and droppable components. Draggables and droppables, as we saw, are very closely related and have been designed to be used with each other. Each supporting and building upon the other enables us to create advanced and highly interactive interfaces.

We've covered a lot of material in this chapter, so let's recap on what we learned. We saw that the draggable behavior can be added to any elment on the page with zero configuration. There may be implementations where this is acceptable, but usually we'll want to use one or more of the component's extensive range of configurable properties.

In the second part of this chapter, we saw that the droppables class allows us to easily define areas on the page that draggables can be dropped onto, and can react to things being dropped on them. We can also make use of a smaller range of configurable droppable properties to implement more advanced droppable behavior.

Both components feature an effective event model for hooking into the interesting moments of any drag and drop interaction. We also saw that each component has a simple set of methods for enabling or disabling drag or drop, and also a `destroy` method for removing the functionality (but not the underlying elements) from the page.

# 9
# Resizing

In this chapter, we continue our journey through the low-level interaction helpers by paying a visit to the resizable component. We have already seen it in action when we looked at the dialog widget earlier in the book. This time, we're going to focus directly on this utility instead of looking at it incidentally. The dialog however is a perfect example of how useful the resizable component can be in a real-world implementation.

This is an interesting component because it's an interaction component as opposed to a full widget, and yet it still automatically adds DOM elements to the page. This bridges the gap between high-level widget and low-level interaction helper nicely.

The resizable is a flexible component that can be used with a wide range of different elements. For example, `<textarea>` elements that may have different amounts of user-entered text in them could styled so the `<textarea>` would be quite small initially. Users could then resize it as they saw fit depending on how much text they entered into it.

Throughout the examples in this chapter, we'll mostly be using simple `<div>` elements so that the focus remains on the component and not on the underlying HTML. We will also look at some brief examples using `<img>` and `<textarea>` elements towards the end of the chapter.

In this chapter, we'll be looking at the following aspects of the component:

- Implementing basic resizability
- Skinning the resizable
- The configurable properties available for use
- Specifying which resize handles to add
- Managing the resizable's minimum and maximum sizes
- The role of resize helpers and ghosts
- A look at the built-in resize animations
- How to react to resize events

The resizables component works well with other components and is very often used in conjunction with draggables. However, while you can easily make draggable components resizable (think dialog), the two classes are in no way related.

# A basic resizable

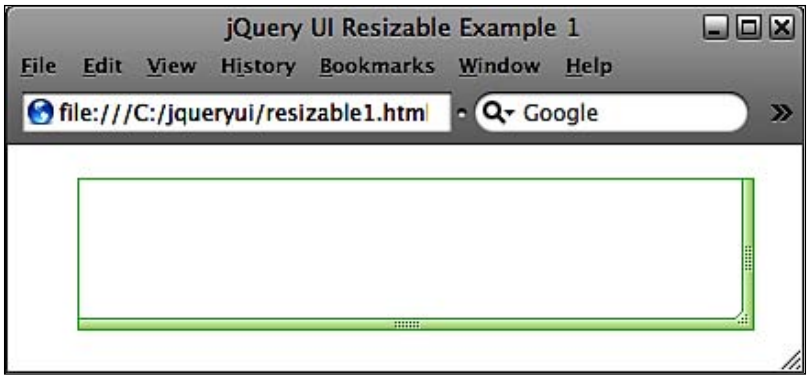
Let's implement the basic resizable so we can see just how easy making elements resizable is when you use jQuery UI as the driving force behind your pages. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/resize.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 1</title>
  </head>
  <body>
    <div class="resize"></div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //make specified element resizable
        $(".resize").resizable();
      });
    </script>
  </body>
</html>
```

Save this as `resizable1.html`. The basic constructor method, used with no arguments for the default implementation, uses the same simplified syntax as the rest of the library. This requires just one line of specific code for the example to work. Apart from the `flora` skin file, we also use a custom stylesheet to add basic dimension and borders to our `<div>`. Use the following CSS in a new stylesheet:

```
.resize {
  width:200px; height:200px;
  margin:30px 0 0 30px; border:1px solid #66cc00;
}
```

Save this file as `resize.css` in the `styles` folder. We've set the dimension properties because without them the `<div>` will stretch the width of the screen. We've also specified a border that matches the `flora` theme because the default implementation only adds resize handles to the right and bottom sides of the targeted element. The following screenshot shows how our basic page should look after the `<div>` has been resized:



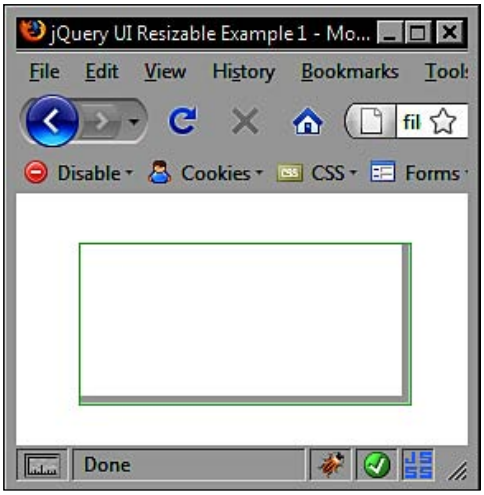

The library files we use in this example are as follows:

- `flora.resizable.css`
- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.resizable.js`

The component automatically adds the three required elements for the drag handles. It even takes care of adding the resize pointers for us when the mouse hovers over one of the sides of the element.

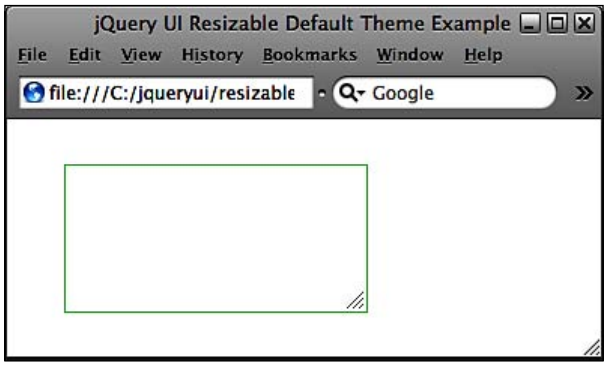
The element can be resized along each axis independently using the side handles. Or, it can be resized along both axis simultaneously using the corner handle. Once again, the library takes care of everything for us.

I also want to mention that although `flora` provides an attractive set of resize handles for us, using the theme is not a mandatory requirement. If an element is made resizable, and the `flora` theme is not specified, the element will automatically be given a light-grey border that can be used to resize the element. If you comment out the link to the `flora` stylesheet in the previous example, you should see the faint grey borders, as in the following screenshot:




Please note that I've darkened the grey borders slightly so that they are clearer in the screenshot. The automatic resize borders are fainter than this when viewed in a browser.

The `default` theme can also be used with the resizable component. With the `default` theme, no borders are added to the resizable element. However, it does get a little corner image added to it to highlight the fact that it is resizable. The image used for the corner is exactly like the image used in the Safari browser, as shown in the following screenshot:

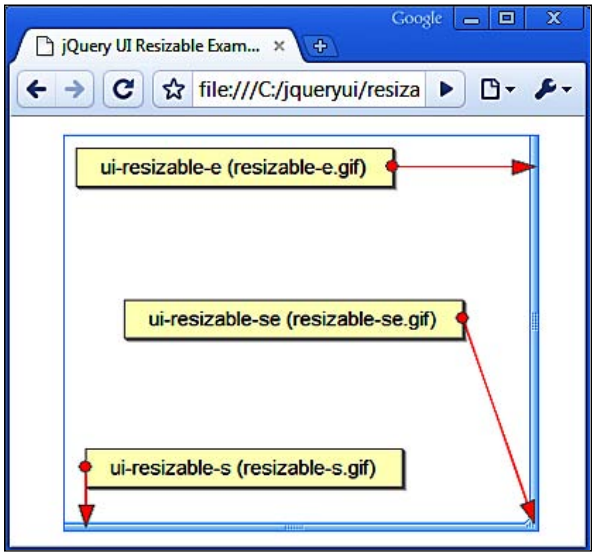


# Skinning the resizable

We looked at creating new images for resize handles earlier in the book when we played with the dialog widget. Let's briefly look at how this can be done once more in this next example. Change `resize.css` so that it appears as follows:

```
.resize {
  width:200px; height:200px;
  margin:30px 0 0 30px;
  border:1px solid #3fa0ff;
}
.ui-resizable-e {
  background:url(../img/resizable/resizable-e.gif) repeat right
center;
}
.ui-resizable-s {
  background:url(../img/resizable/resizable-s.gif) repeat center top;
}
.ui-resizable-se {
  background:url(../img/resizable/resizable-se.gif) repeat;
}
```

Save this as `resizeSkin.css` in the `styles` folder. Link to this new page in `resizable1.html` and resave the file as `resizable2.html`. When we view the new file in a browser, we see that the green theme has been replaced with our blue theme. All it took was three new images and three overriding style rules as you can see here:



The previous screenshot shows the class names of the elements we are targeting with our stylesheet. It also has the names of the images we are using (in brackets) to clarify how we are using the CSS to override style rules for specific elements.

Of course, in this example we are only using three handles. We can choose to use more than three, in which case we would need additional images and style rules to complete the new skin.

# Resizable properties

The following table lists the configurable properties we have at our disposal when working with the resizable component:

| Property | Default Value | Usage |
|---|---|---|
| `animate` | `false` | Animates the resizable element to its new size |
| `animateDuration` | `slow` | Sets the speed of the animation; values can be integers specifying the number of milliseconds, or one of the string values `slow`, `normal`, or `fast` |
| `animateEasing` | `swing` | Adds easing effects to the resize animation |
| `alsoResize` | `false` | Use with a jQuery selector to resize another element when the resizable element is resized |
| `aspectRatio` | `false` | Makes all edges of the resizable the same length at all times, maintaining the aspect ratio of the element. |
| `autoHide` | `false` | Hides the resize handles until the resizable is hovered over with the mouse pointer |
| `cancel` | `input` | Stops specified elements from being resized |
| `containment` | `false` | Constrains the resizable within the boundary of the specified container element |
| `delay` | `0` | Sets a delay in milliseconds from when the pointer is clicked on a resizable handle to when the resizing begins |
| `disableSelection` | `true` | Stops handles and resize helper elements from being selected |
| `distance` | `1` | Sets the number of pixels the mouse pointer must move with the mouse button held down before resizing begins |

| Property | Default Value | Usage |
|---|---|---|
| `ghost` | `false` | Shows a substitute element while the resizing is taking place |
| `grid` | `false` | Accepts an object specifying x and y coordinates to snap the resize to while resizing is taking place |
| `handles` | `{ e, se, s }` | Defines which handles to use for resizing and uses an object specifying the handle names (n, s, e, w, etc) as the keys and jQuery selectors or DOM nodes as the values |
| `helper` | `null` | Enables a helper element which shows the resize while it is in progress and is very similar to, but simpler than, the `ghost` property |
| `knobHandles` | `false` | Uses simple handles instead of image-based handles |
| `maxHeight` | | Sets the maximum height the resizable may be changed to |
| `maxWidth` | | Sets the maximum width the resizable may be set to |
| `minHeight` | | Sets the minimum height the resizable may be changed to |
| `minWidth` | | Sets the minimum width the resizable may be set to |
| `preserveCursor` | `true` | Shows the resizing mouse pointer while hovering over a resize handle |
| `preventDefault` | `true` | Prevents Safari's automatic `<textarea>` resizing feature |
| `proportionallyResize` | `false` | Accepts an array of jQuery selectors or DOM nodes that should be proportionally resized when the resizable is resized |
| `transparent` | `false` | No resize handles are shown either before, during, or after an interaction |

# Configuring resize handles

Thanks to the `handles` configuration property, specifying which handles we would like displayed is exceptionally easy. In a new file in your text editor, add the following code:

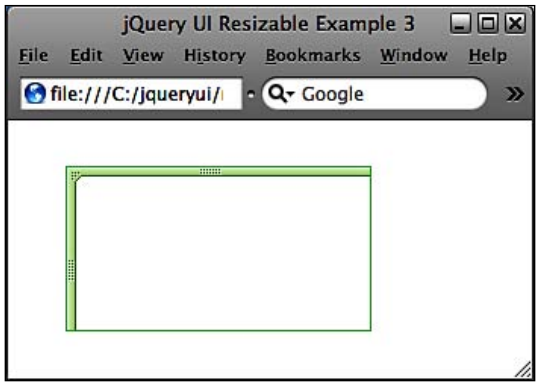
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/resize.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 3</title>
  </head>
  <body>
    <div class="resize">
      <div class="ui-resizable-handle ui-resizable-w"></div>
      <div class="ui-resizable-handle ui-resizable-n"></div>
      <div class="ui-resizable-handle ui-resizable-nw"></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create config object
        var resizeOpts = {
          handles:{
            w: ".ui-resizable-w",
            n: ".ui-resizable-n",
            nw: ".ui-resizable-nw"
          }
        }
        //make specified element resizable
        $(".resize").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this as `resizable3.html`. The underlying HTML for this example has changed with the use of the `handles` property. When using this property, it is essential to supply elements to be used as the drag handles. We cannot just leave this up to the component to sort out.

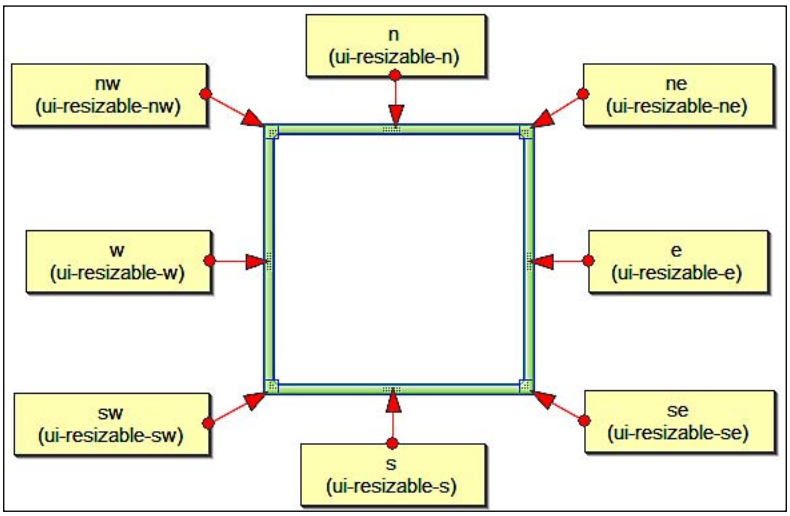
Within our resize `<div>`, we have therefore added three new `<div>` elements which will be transformed by the component into the resize handles. Each has been given a class of `ui-resize-handle` and the appropriate compass-point class name.

The value that each of these properties takes in this example is a jQuery class selector. This will match class name of the element that is to be used as the handle, although they can also accept DOM nodes. The following screenshot shows how the resizable should now appear:




We've used the same class names as those used by the component, so our handles will automatically pick up the `flora` styling. The `handles` property itself expects a literal object consisting of one or more of the compass-point properties. These are very similar to the compass-point properties we used with the dialog widget back in chapter 4.

The value of each property should match the class name of the element which is to become a handle. The following image shows which part of the resizable matches each compass-point property. The class names used to pick up the `flora` styling are also shown in brackets:





**The all Property**

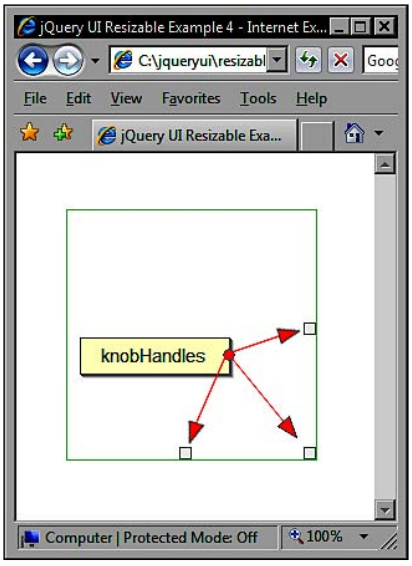
Apart from an object of compass-points and class names, we can also supply the string `all` to the `handles` property. This will add resize handles to all edges of the resizable element. Note that you don't need to use underlying HTML elements in your markup when using the `all` property in conjunction with the `flora` stylesheet (see `resizable3allHandles.html` for clarification).

There are a couple of additional properties that relate to resize handles and how they are displayed. We'll look at these two properties next. Remove the three handle `<div>` elements from the resizable `<div>` and change the configuration object in `resizable3.html` so that it appears as follows:

```
//create config object
var resizeOpts = {
  knobHandles: true,
  autoHide: true
};
```

Save this version as `resizable4.html`. We simply set both the `knobHandles` and `autoHide` properties to `true` in this example. Let's explore what each property does. Setting `knobHandles` to `true` enables the use of simple square resize handles instead of the more complex `flora` handles. Setting `autoHide` to `true` hides the resize handles until the mouse pointer moves onto the resizable element.

The following screenshot shows the small square resize handles given to the component by the `knobHandles` property:



# Defining size limits

So far, our resizable has been devoid of any content of its own. In the following example, we can add some simple layout text to enhance our learning experience. Change `resizable4.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/resize.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 5</title>
  </head>
  <body>
    <div class="resize">
      <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Suspendisse auctor ligula vel odio. Nam et sem vitae nibh convallis
euismod. Aenean vitae urna quis augue adipiscing hendrerit. Nam
faucibus. Phasellus eros. Ut bibendum eros at nibh. Sed erat. Aenean
id enim vitae leo aliquet rutrum. Mauris fringilla euismod orci.
Morbi tempus purus eget felis. Sed dui eros, tempor id, iaculis vel,
pretium eget, nunc. Pellentesque vehicula tincidunt arcu. Ut a velit.
In dapibus, lacus vitae lobortis dictum, libero pede venenatis magna,
eu sagittis nunc erat sed ante. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Phasellus est
dolor, mollis congue, ullamcorper eu, auctor ut, augue.</p>
    </div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
     //function to execute when doc ready
     $(function() {
       //create config object
       var resizeOpts = {
        maxWidth: 500,
        maxHeight: 500,
        minWidth: 100,
        minHeight: 100
```

```
        }
         //make specified element resizable
         $(".resize").resizable(resizeOpts);
       });
    </script>
   </body>
</html>
```

Save this as `resizable5.html`. This time, the configuration object uses the dimension boundary properties to specify minimum and maximum heights and widths that the resizable may be adjusted to. These properties take simple integers as their values, which are then converted to pixels by the component.

There is one thing you may notice now that our resizable has content in it. If you shrink the resizable element so that it is smaller than the content it contains, the content itself is still visible and simply overlaps the boundaries of the resizable. We can fix this easily enough by adding the style rule `overflow:hidden` to our stylesheet.

This is really the only sensible value to give the `overflow` style attribute when we use the resizable component. Setting `overflow` to `none` or `automatic` does nothing in this example, and setting it to `scroll` adds the highly unattractive standard OS scrollbars to the resizable element.

You should note however that Internet Explorer will break the resizable if we use `overflow:hidden` and there is overflow content. We could reduce the amount of content within the resizable or make the resizable bigger by default to overcome this difficulty. Try the page out and you should notice the resizable will now keep the dimensional properties that we specified in our configuration object.

# Resize ghosts

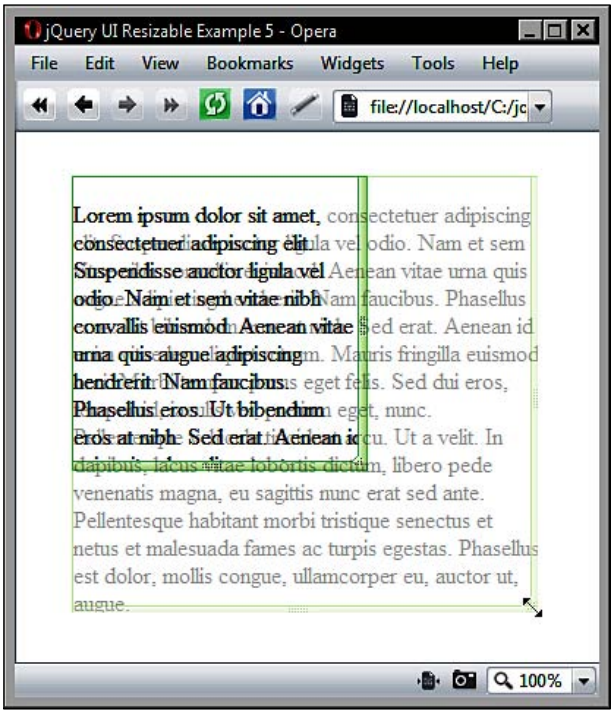
Ghost elements are very similar to the proxy element we used when we looked at the draggable and droppable components in the previous chapter. They can also play a part in the resizables component as well. A ghost element can be enabled with the configuration of just one property. Let's see how this is done. Change `resizable5.html` to this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/resize.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 6</title>
  </head>
  <body>
    <div class="resize">
      <p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Suspendisse auctor ligula vel odio. Nam et sem vitae nibh convallis
euismod. Aenean vitae urna quis augue adipiscing hendrerit. Nam
faucibus. Phasellus eros. Ut bibendum eros at nibh. Sed erat. Aenean
id enim vitae leo aliquet rutrum. Mauris fringilla euismod orci.
Morbi tempus purus eget felis. Sed dui eros, tempor id, iaculis vel,
pretium eget, nunc. Pellentesque vehicula tincidunt arcu. Ut a velit.
In dapibus, lacus vitae lobortis dictum, libero pede venenatis magna,
eu sagittis nunc erat sed ante. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Phasellus est
dolor, mollis congue, ullamcorper eu, auctor ut, augue.</p>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
         //create config object
        var resizeOpts = {
          ghost: "true"
                }
        //make specified element resizable
        $(".resize").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this file as `resizable6.html`. All that is needed to enable a resize ghost is to set the `ghost` property to `true`. No additional underlying HTML or styling is required. Everything is handled by the component for us.

The next screenshot shows how the resizable ghost will appear while it is visible:




# Constraining the resize and maintaining ratio

The component makes it easy to ensure that a resized element is constrained to its container element. This is great if we have other content on the page that we don't want moving around all over the page during a resize interaction.

We can also ensure that the specified element is resized symmetrically along both the x and y axis. This is known as maintaining its aspect ratio and makes the component very useful when resizing images. We can make use of both properties in our next example. Change `resizable6.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
resizeContainer.css">
```

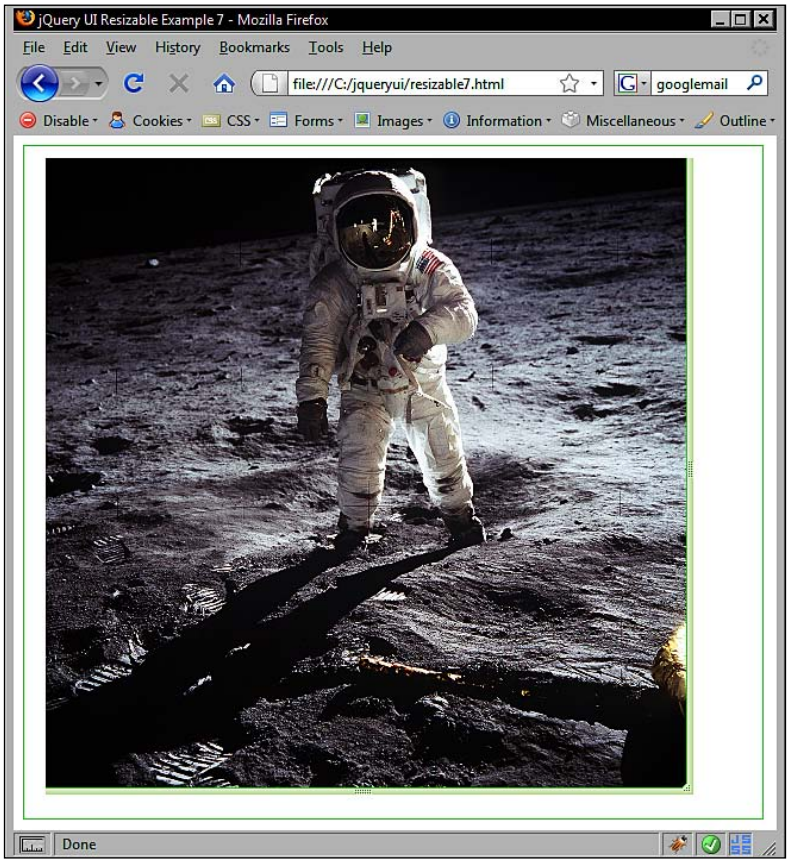
```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 7</title>
  </head>
  <body>
    <div class="container">
      <img id="resize" src="img/resizable/moon.jpg" alt="Moon
Landing">
    </div>
<script type="text/javascript" src="jqueryui1.6rc2/jquery-1.2.6.js">
</script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //create config object
        var resizeOpts = {
          containment:".container",
          aspectRatio: true
        }
        //make specified element resizable
        $("#resize").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this as, you've guessed it, `resizable7.html`. Before we look at the properties used in this example, I should point out that we also use some different CSS for this example. In a new file in your text editor, add the following code:

```
.container {
  width:600px; height:600px; border:1px solid #66cc00;
  padding:10px;
}
#resize { width:300px; height:300px; }
```

Save this as `resizeContainer.css` in the `styles` folder. Now, about those configuration properties. The `containment` property allows us to specify a container for the resizable which will limit how large the resizable can be made, forcing it to stay within its boundaries. We specify a jQuery selector as the value of this property.

The other property that we've used in this example is `aspectRatio`, which makes each side of the resizable stay the same size. This ensures our resizable element will always be a square as opposed to a rectangle. When you run this page in your browser, you should see that the original aspect ratio of the image is preserved, and that the image cannot be made bigger than its container:




# Resizable animations

The resizable API exposes three properties related to animations, which are the `animate`, `animateDuration`, and `animateEasing` properties. By default, animations are switched off. However, we can easily enable them to see how they enhance the component. Create the following new page in your text editor:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/
resizeAnimate.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 8</title>
  </head>
  <body>
    <textarea id="resize"></textarea>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var resizeOpts = {
          helper: "proxy",
          knobHandles: true,
          animate: true,
          animateDuration: "fast"
        };
        //make specified element resizable
        $("#resize").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this as `resizable8.html`. This example is based on a `<textarea>` as this can be a useful element to make resizable. The configuration object we use in this example starts with the `helper` property. When using animations, the resizable element is not resized until after the resize interaction has ended.

The `helper` property is useful to show the user the new size of the resizable while the resize is taking place. The value we give to the `helper` property becomes the class name that is applied to the helper element, which we can use to target with some minimal styles. In principle, the resizable helper is very similar to the ghost, but it does not show the inner content of the resizable.

We use the `knobHandles` property again, simply for the purpose in this example that it looks better than the `flora` resize handles. We'll also need some custom styling for them, which we'll add in a moment.

All we need to do to enable animation is set the `animate` property to `true`. That's it, no further configuration is required. Another option we have is to set the speed of the animation, which we have done in this example, by supplying the `animateDuration` property. This can either be an integer to represent the number of milliseconds the animation can last for, or using one of the strings `slow`, `normal`, or `fast`.

# Resizable callbacks

Like other components of the library, resizable defines a selection of custom events and allows us to easily execute functions when these events occur. This makes the most of interactions between your visitors and the elements on your pages. Resizable defines the following callback properties:

| Property | Triggered |
|---|---|
| `resize` | When the resizable is in the process of being resized |
| `start` | When the resize interaction begins |
| `stop` | When the resize interaction ends |

Hooking into these custom methods is just as easy for resizables as it has been for the other components of the library we have looked at. Let's explore a basic example to highlight this fact. Create the following new page in your text editor:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
    <link rel="stylesheet" type="text/css" href="styles/resize.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Example 9</title>
  </head>
  <body>
    <div class="resize">
      <p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
      Suspendisse auctor ligula vel odio. Nam et sem vitae nibh
      convallis euismod.</p>
    </div>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var resizeOpts = {
          stop: reportNewSize
        }
        //display new size of resizable
        function reportNewSize() {
          //create and display the tip
          $("<div>").addClass("tip").text("The resizable is now " +
$(this).height() + " pixels high, and " + $(this).width() + " pixels
wide").css({
            border: "2px solid #66cc00",
            fontSize: "80%",
            fontWeight: "bold",
            position: "absolute",
            display: "none",
            left: 38,
            marginTop: 5,
            width: $(this).width() - 2
          }).appendTo("body").fadeIn("slow", goAway);
          //hide the tip
          function goAway() {
            setTimeout("$('.tip').fadeOut('slow')", 2000);
          }
        }
        //make specified element resizable
        $(".resize").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this as `resizable9.html`. We use the `stop` property to specify a callback function that will be executed as soon as the resize interaction stops. Our callback simply creates a new `<div>` element and adds a string of text to it. It then sets some of the new element's CSS properties before appending it to the page after the resizable and calling the standard jQuery `fadeIn()` method.

We can also easily use a second callback function, called at the end of the `fadeIn` effect, which hides the new `<div>` after a specified length of time. The following screenshot shows how our page looks before the `<div>` fades away:




Like the other library components, these callbacks can automatically receive up to two arguments which are the event object, and an object containing useful properties of the resizable.

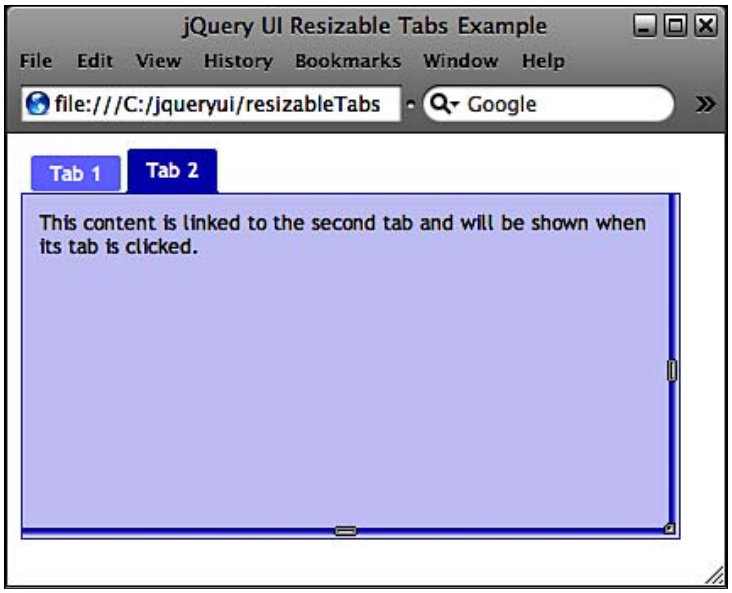
The second object has two properties we can make use of. The `options` property, which gives you access to the options used to initialize the resizable, and the `axis` property, which tells us which handle was dragged. We didn't need to use either of these properties in the last example however, so referred to the `$(this)` object instead.

# Resizable methods

This component comes with the three basic methods found with all of the interaction components of the library, namely the `destroy`, `disable`, and `enable` methods. These work and are used in the same way as the methods by the same names that come with the other interaction components. Therefore, we won't be looking at these in any great detail in this chapter.

# Fun with resizable

For our final resizable example, let's look at combining this component with one of the widgets that we looked at in a previous chapter. This will help us see how compatible this component is with the rest of the library. We'll be working with the tabs component in the following example. The following screenshot shows the page we will end up with:




In your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.resizable.css">
```

```
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
resizableTabsTheme.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Resizable Tabs Example</title>
  </head>
  <body>
    <ul id="myTabs">
      <li><a href="#0"><span>Tab 1</span></a></li>
      <li><a href="#1"><span>Tab 2</span></a></li>
    </ul>
    <div class="tab" id="0">This is the content panel linked to the
first tab, it is shown by default.</div>
    <div class="tab" id="1">This content is linked to the second tab
and will be shown when its tab is clicked.</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.resizable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
      //define function to be executed on document ready
      $(function(){
        //set initial tab size
        var newHeight = 100;
        var newWidth = 300;
        var tabOpts = {
          show: setSize
        }
        function setSize(e, ui) {
          //set the dimensions of the tab
          var panel = ui.panel;
          $(panel).height(newHeight).width(newWidth);
        }
        //create the tabs
        var tabs = $("#myTabs").tabs(tabOpts);
        //define config object for resizeable
        var resizeOpts = {
          autoHide: true,
          stop: resizeSibling
        };
        //resize the other tab at the same time
```

```
        function resizeSibling() {
          //get the new dimensions
          newHeight = $(this).height();
          newWidth = $(this).width();
        }
        //make tabs resizable
        $(".tab").resizable(resizeOpts);
      });
    </script>
  </body>
</html>
```

Save this as `resizableTabs.html`. We also link to a new stylesheet for this example. It's similar to those used in previous tab and resizable examples and contains the following code:

```
.ui-tabs-panel {
  border:1px solid #0000cc;
  background:#d8d8f7;
}
.ui-tabs-nav a, .ui-tabs-nav a span {
  background:url(../img/tab-sprite.gif) no-repeat;
}
.ui-tabs-nav a {
  background-position:100% 0%;
}
.ui-resizable-e {
  background:url(../img/resizable/tabResizable-e.gif) repeat right
center;
  width:7px;
}
.ui-resizable-s {
  background:url(../img/resizable/tabResizable-s.gif) repeat center
top; height:7px;
}
.ui-resizable-se {
  background:url(../img/resizable/tabResizable-se.gif) repeat 0%;
}
```

This can be saved as `resizableTabsTheme.css` in the `styles` folder. Making the tabs widget resizable is extremely easy and only requires calling the `resizable` method on tab's underlying `<ul>`.

We're using two configuration objects in this example. One object for each component. Apart from setting the `autoHide` property for the resizable in our configuration object, we also define a function that should be called when the `stop` event occurs. The function executed whenever a tab is shown is to set the dimensions of the tab panel that has just been shown.

Because this function will be called when the page loads, as well as on each subsequent tab display, we also specify initial values that are passed to the `width` and `height` jQuery methods.

The second function, which is executed whenever a resize occurs, simply gets the new size of the tab that has been resized and saves the new `width` and `height` values to our two variables for later use (such as whenever a tab is shown). Together, these functions allow you to resize a tab, and have all tabs assume the new size.

# Summary

In this chapter we covered resizables. This is a component which allows us to easily resize any on-screen element. It dynamically adds resize handles to the specified sides of the target element and handles all of the tricky DHTML resizing for us, neatly encapsulating the behaviour into a compact, easy-to-use class.

We first looked at the different theming options available when using resizable, and how easy it is to create our own theme by overriding the original styling of the `flora` or `default` themes.

We then looked at some of the configurable properties we can use with the widget, such as how to specify which handles to add to the resizable, and how the minimum and maximum sizes of the element can be limited.

We briefly looked at how to maintain an image's aspect ratio while it is being resized. We also explored how to use ghosts, helpers, and animations to improve the usability and appearance of the resizable component.

We looked at the event model exposed by the component's API and how we can react to elements being resized in an easy and effective way. Our final example explored resizable's compatibility with other widgets in the library.

# 10
# Selecting

The selectables component allows you to define a series of elements that can be 'chosen' by dragging a selection square around them or by clicking them, as if they were files in Windows Explorer (or Finder on the Mac). In this way, elements on the page can be treated as file-like objects, allowing either single or groups of elements to be selected.

A selection square has been a standard part of modern operating systems for a long time. For example, if you wanted to select some of the icons on your desktop, you could hold the mouse button down on a blank part of the desktop and drag a square around the icons you wanted to select.

The selectables interaction helper adds this same functionality to our web pages, which could be very useful in a variety of situations. This is yet another example of how the web is increasingly becoming less distinct from the desktop as an application platform.

Topics that will be covered in this section include:

- Creating the default implementation
- How selectable class names reflect the state of selectables
- Filtering selectable elements
- Working with selectable's built-in callback functions
- A look at selectable's methods

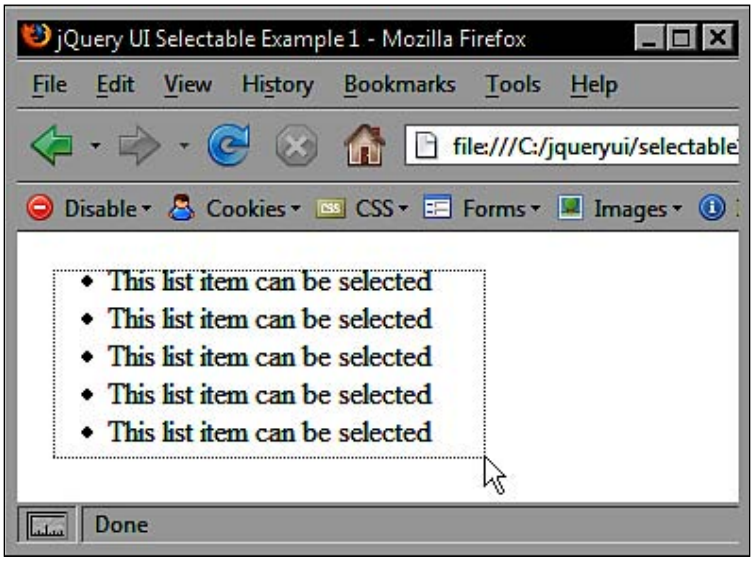
# Basic implementation

A demonstration that you can play with will tell you more about the functionality provided by this library component than merely reading about it. The first thing we should do is invoke the default implementation to get a glimpse at the effects of this component. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Selectable Example 1</title>
  </head>
  <body>
    <ul id="selectables">
      <li>This list item can be selected</li>
      <li>This list item can be selected</li>
      <li>This list item can be selected</li>
      <li>This list item can be selected</li>
      <li>This list item can be selected</li>
    </ul>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //make specified elements selectable
        $("#selectables").selectable();
      });
    </script>
  </body>
</html>
```

Save this as `selectable1.html` and run it in a browser. You should observe that you can drag a selection square around one or more of the list items. The list items don't do anything once they are selected of course, as this is only the default implementation. We simply call the `selectable` constructor method on the parent list element and then all of its child `<li>` elements are made selectable.

Note that there is no `default` or `flora` styling associated with the selectables component. Other default behavior includes clicking on individual elements causes only them to be selected and clicking outside of the selected elements will deselect them. Holding down the *Ctrl* key while clicking will enable multi-select. The following screenshot shows the selected square enclosing the list items:




The minimum set of library files we need for a selectable implementation is:

- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.selectable.js`

Apart from building selectables from list items, we can also build them from other elements, such as a collection of `<div>` elements:

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
selectable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Selectable Example 2</title>
  </head>
  <body>
```

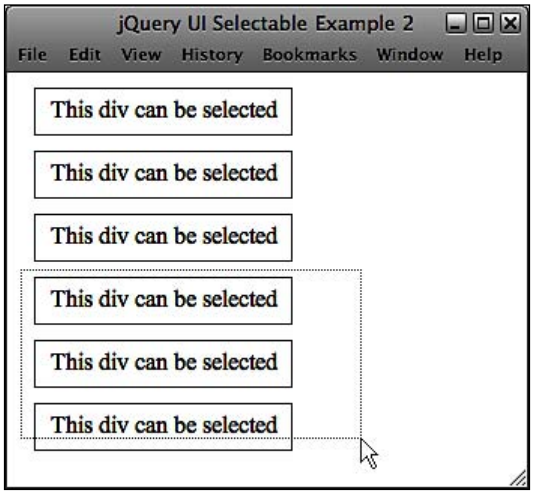
```
    <div id="selectables">
      <div>This div can be selected</div>
      <div>This div can be selected</div>
      <div>This div can be selected</div>
      <div>This div can be selected</div>
      <div>This div can be selected</div>
      <div>This div can be selected</div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //make specified elements selectable
        $("#selectables").selectable();
      });
    </script>
  </body>
</html>
```

Save this as `selectable2.html`. Everything is essentially the same as before. We're just basing the example on different elements, using `<div>` instead of `<li>`. However, due to the nature of these elements, we should add a little basic styling so that we can see what we're working with.

In a new file in your text editor, add the following code:

```
#selectables div {
  width:160px; height:25px;
  padding:5px 0 0 10px; margin:10px 0 0 10px;
  border:1px solid #000;
}
```

Save this as `selectable.css` in your `styles` folder. It's not much, but it helps to clarify the individual selectables in the example, as shown in the following screenshot:



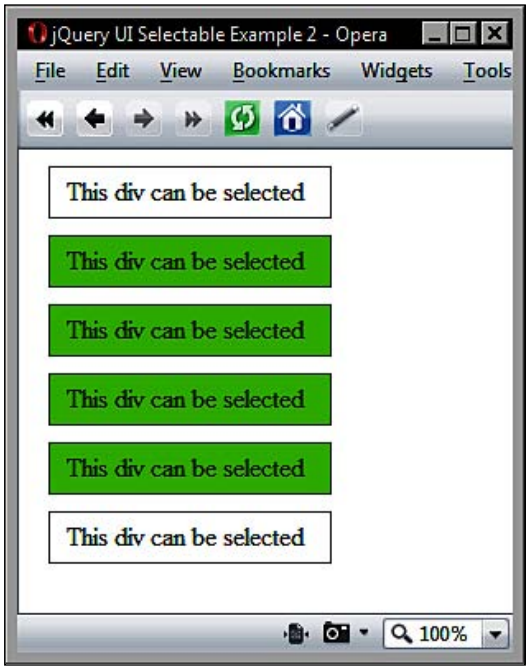

# Selectee class names

The elements which are made selectable are all initially given the class `ui-selectee`. While the selecting square is actually around selectable elements, they are given the class `ui-selecting`. Once the select interaction ends, any selectables that have been selected are given the class `ui-selected`. A previously selected element that is not part of the current selection is given the class `ui-unselecting`.

The component also makes it very easy to add custom styling to show when elements are either in the process of being selected or have been selected. Let's add some additional styling now to reflect the *selecting* and *selected* states. Add the following new selectors and rules to `selectable.css`:

```
#selectables div.ui-selecting {
  border:1px solid #66CC00;
}
#selectables div.ui-selected {
  background:#66CC00;
}
```

With the addition of this simple CSS, we can add visual cues to elements which are part of the current selection, both during and following a select interaction. The following screenshot shows that some elements have been selected:




# Configurable properties of the selectable class

The selectable class is quite compact, with relatively few configurable properties compared to the other interaction helpers. The following properties are available for configuration:

| Property | Default Value | Usage |
|---|---|---|
| `autoRefresh` | `true` | Automatically refreshes the size and position of each selectable at the start of a select interaction |
| `filter` | `"*"` | Used to specify child elements to make selectable |

# Filtering selectables

There may be situations when we don't want to allow all of the elements within the targeted container selectable. In this situation, we can easily make use of the `filter` property to nominate specific elements, based on a CSS selector, that we want selecting to be enabled on. Change `selectable2.html` so that it appears as follows:

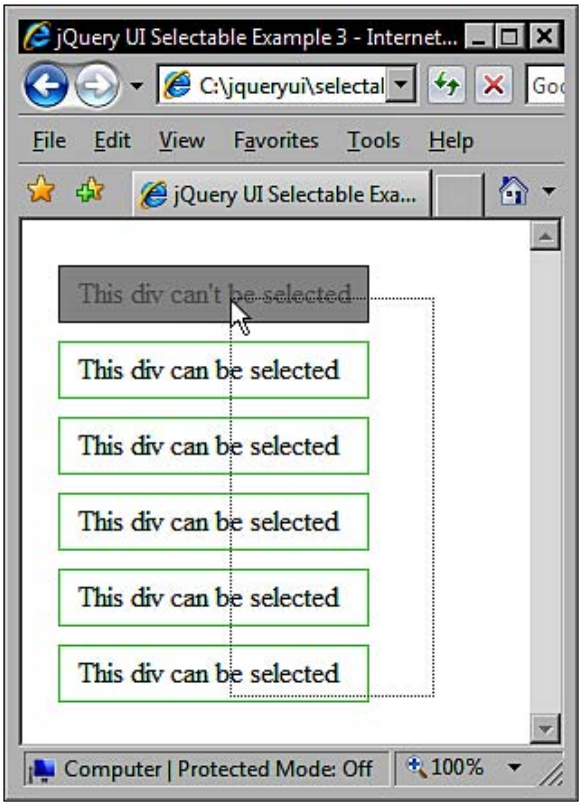
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
selectableFiltered.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Selectable Example 3</title>
  </head>
  <body>
    <div id="selectables">
      <div class="unselectable">This div can't be selected</div>
      <div class="selectable">This div can be selected</div>
      <div class="selectable">This div can be selected</div>
      <div class="selectable">This div can be selected</div>
      <div class="selectable">This div can be selected</div>
      <div class="selectable">This div can be selected</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var selectableObj = {
           filter: ".selectable"
        }
        //make specified elements selectable
        $("#selectables").selectable(selectableObj);
      });
    </script>
  </body>
</html>
```

Save this version as `selectable3.html`. In the underlying mark-up, we have given different class names to different elements. This is based on whether we want them to be selectable or not. In the JavaScript, we define a configuration object containing the `filter` property. The value of this property is the class selector of the elements that we want to be selectable.

We also used a new stylesheet in this example to give the unselectable elements their own styling. This new stylesheet is the same as the previous stylesheet with the addition of the following selector and rules:

```
.unselectable { background-color:#999999; color:#666666; }
```

The new stylesheet can be saved as `selectableFiltered.css`. The following screenshot shows how the page should look:



# Selectable callbacks

In addition to the two standard properties of the selectable API, there are also a series of properties that can be used to specify executable callback functions at specific points during a select interaction. These properties are as follows:

| Property | Triggered When |
|---|---|
| `selected` | The select interaction ends and each element added to the selection triggers the callback |
| `selecting` | Each selected element triggers the callback during the select interaction |
| `start` | A select interaction begins |
| `stop` | This is fired once regardless of the number of items selected as the select interaction ends |
| `unselected` | Any elements that are part of the selectable but are not selected during the interaction will fire this callback |
| `unselecting` | Unselected elements will fire this during the select interaction |

Like the draggable and droppable components that we looked at earlier, selecting really only becomes useful when something happens to the elements once they have been selected. Let's put some of these callbacks to work so that we can appreciate their use. Change `selectable3.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
selectable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Selectable Example 4</title>
  </head>
  <body>
    <div id="selectables">
      <div id="selectabl1" class="selectable">This div can be
selected</div>
      <div id="selectabl2" class="selectable">This div can be
selected</div>
      <div id="selectabl3" class="selectable">This div can be
selected</div>
      <div id="selectabl4" class="selectable">This div can be
selected</div>
      <div id="selectabl5" class="selectable">This div can be
selected</div>
```

```
      <div id="selectabl6" class="selectable">This div can be
selected</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var selectableObj = {
          selected: function(e, ui) {
            $("#" + ui.selected.id).text("I have been selected!");
          },
          unselected: function(e, ui) {
            $("#" + ui.unselected.id).text("This div can be
selected");
          },
          start: function(e) {
            $("<div>").attr("id", "tip").text("Drag the lasso around
elements, or click to select").css({
              position:"absolute", backgroundColor:"#ffffcc",
              border:"1px solid #3366ff", width:"310px",
              height:"20px", textAlign:"center",
              left:e.pageX, top:e.pageY - 30
            }).appendTo($("body"));
          },
          stop: function() {
            $("#tip").fadeOut();
          }
        }
        //make specified elements selectable
        $("#selectables").selectable(selectableObj);
      });
    </script>
  </body>
</html>
```

Save this as `selectable4.html`. To the HTML elements, we've added `id` attributes so that we can easily target specific elements. In the `<script>`, we've added anonymous functions to the `selected`, `unselected`, `start`, and `stop` properties. These will be executed at the appropriate times during an interaction.

As with other components, these functions are automatically passed two objects. The first is the original browser event object and the other represents the selectable element. However, not all callbacks can successfully work with the second object – `start` and `stop` for example.

When a `<div>` is selected, we change its inner text to reflect the selection using the `selected` anonymous function. We are able to get the `id` of the element that has been selected using the `selected.id` property of the second object that is automatically passed to our function. When an element is unselected, we set the text back to its original value using the same technique.

We can also alter the inner text of any selectable that hasn't been selected using the `unselected` function. This could be useful for letting our visitors know that the element could be selected if they wanted to include it in the selection.

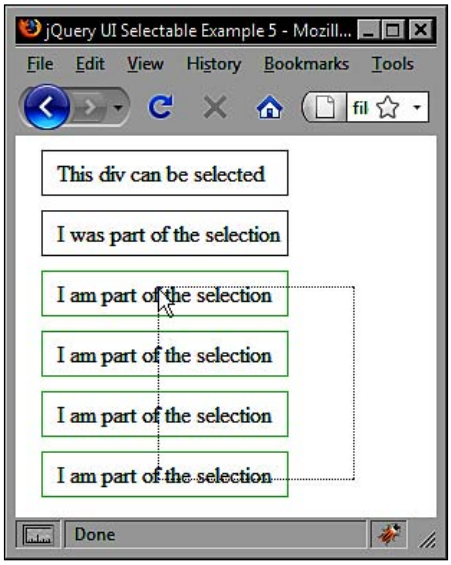
At the start of any interaction, we create a little tool tip that is appended to the `<body>` of the page, slightly offset from the mouse pointer, using the `start` anonymous function. We can get the pointer coordinates using the `e` (event) object, which is passed as the first argument to our callbacks. At the end of the selection, we then remove the tool tip using the `stop` property..

The `selecting` and `unselecting` callback properties work in exactly the same way. For an example of how they work, we could use the following `<script>` block:

```
<script type="text/javascript">
  //function to execute when doc ready
  $(function() {
    //define config object
    var selectableObj = {
      selecting: function(e, ui) {
        $("#" + ui.selecting.id).text("I am part of the selection");
      },
      unselecting: function(e, ui) {
        $("#" + ui.unselecting.id).text("I was part of the
selection");
        }
    }
    //make specified elements selectable
    $("#selectables").selectable(selectableObj);
  });
</script>
```

Save this as `selectable5.html`. This time we use the `selecting` and `unselecting` properties to specify anonymous functions, which again, change the inner text of the elements at certain times during an interaction.

We do the same type of thing as before using the same techniques. This time, we're just using different callbacks and different properties of the objects passed to them. The effects of these callbacks are shown in the following screenshot:




The second object passed to any of the selectable callbacks contains a property relating to the type of custom event. For example, the `selected` callback receives an object with a `selected` property which can be used to gain information about the element that was added to the selection. All callbacks have a matching property that can be used in this way.

# Selectable methods

The methods that we can use to control the selectables component from our code are similar to the methods found in the other interaction components and follow the same pattern of usage. These are listed in the following table:

| Method | Usage |
|---|---|
| `disable` | Disables selectable functionality |
| `enable` | Re-enables selectable functionality |
| `refresh` | Manually refreshes the positions and sizes of selectables and is used when `autoRefresh` is set to `false`. |
| `toggle` | Toggles the enabled and disabled states of selectables |
| `destroy` | Permanently removes selectable functionality |

Two new methods that are unique to this component are the `toggle` and `refresh` methods. When the `autoRefresh` property is set to `false`, the `refresh` method can be used to manually perform a refresh at certain times.

Setting the `autoRefresh` property to `false` can yield performance gains when there are many selectables on the page. However, there will still be times when you will need to refresh the size and positions of the selectables, which is exactly what the `refresh` method does.

The `toggle` method allows you to easily switch between *enabled* and *disabled* states, without having to have separate code routines for the two states, and without having to do any kind of state detection.

If the selectables are currently enabled, `toggle` will disable them. If they are currently disabled, `toggle` will enable them. Using this method is child's play, but we haven't come across it in any of the other components. So, let's take a quick look at it in the wild. Create the following new page in your text editor:

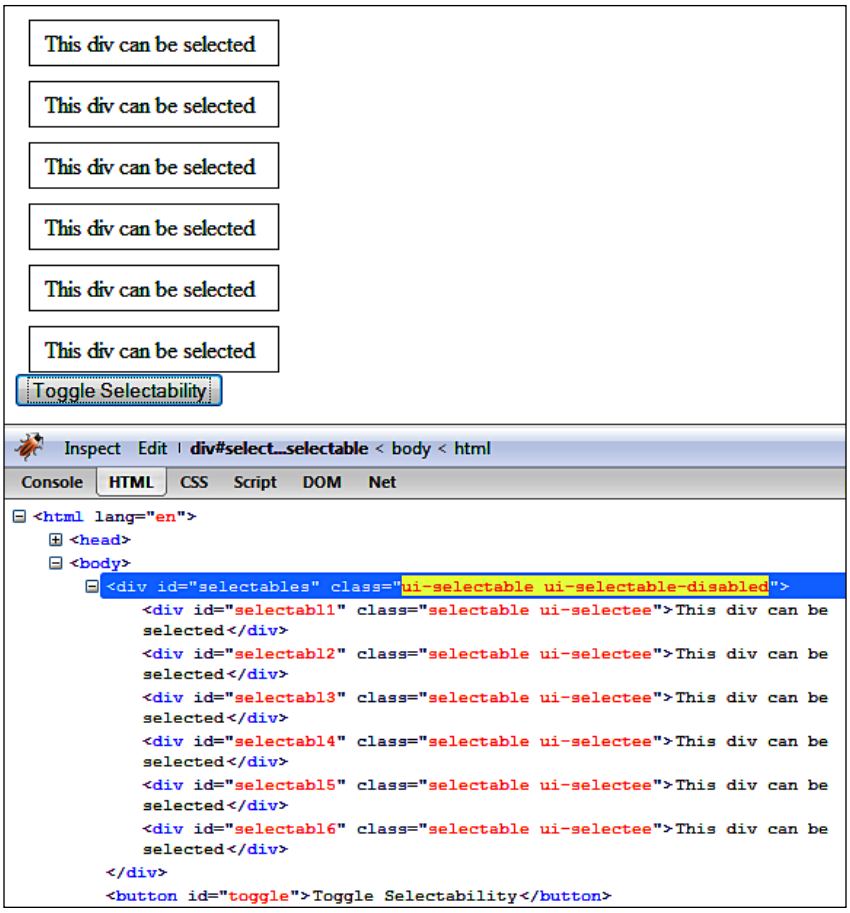
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
selectable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Selectable Example 6</title>
  </head>
  <body>
    <div id="selectables">
      <div id="selectabl1" class="selectable">This div can be
selected</div>
      <div id="selectabl2" class="selectable">This div can be
selected</div>
      <div id="selectabl3" class="selectable">This div can be
selected</div>
      <div id="selectabl4" class="selectable">This div can be
selected</div>
      <div id="selectabl5" class="selectable">This div can be
selected</div>
      <div id="selectabl6" class="selectable">This div can be
selected</div>
    </div>
    <button id="toggle">Toggle Selectability</button>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //make specified elements selectable
        $("#selectables").selectable();

        //define click handler for button
        $("#toggle").click(function() {

          //toggle selectability
          $("#selectables").selectable("toggle");
        });
      });
    </script>
  </body>
</html>
```

Save this as `selectable6.html`. The page contains a new `<button>` element, which enables or disables selectability depending on its current state. After making the parent `<div>` selectable in the normal way, we then define a click handler for the `<button>`. Within this click handler, we simply call the `toggle` method.

At this stage, there will be no visual indication that anything has happened when we click the `<button>`. Although if you use Firebug, you can see that the class name attached to the outer parent of the selectables changes depending on its state.

When the `<button>` is clicked for the first time, the `<div>` is given the additional class name called `ui-selectable-disabled`. You can see this in the following screenshot:




If the `<button>` is clicked a second time, the extra class name is removed. To make it more obvious that the selectables have been disabled (or enabled again), we can use this additional class name to add some alternative styling to identify when selectability is disabled. In a new file in your text editor, add the following selector and rules:

```
.ui-selectable-disabled .selectable {
  border:1px solid #666666; background-color:#cccccc;
  color:#999999;
}
```

Save this as `selectableToggle.css` in the `styles` folder. In `selectable6.html`, add our new stylesheet to the `<head>` of the page:

```
<link rel="stylesheet" type="text/css" href="styles/selectableToggle.
css">
```

Resave the page as `selectable7.html`. Now when the `<button>` is clicked, our new style rules are applied and it becomes easier to see that something has happened, as in the following screenshot:

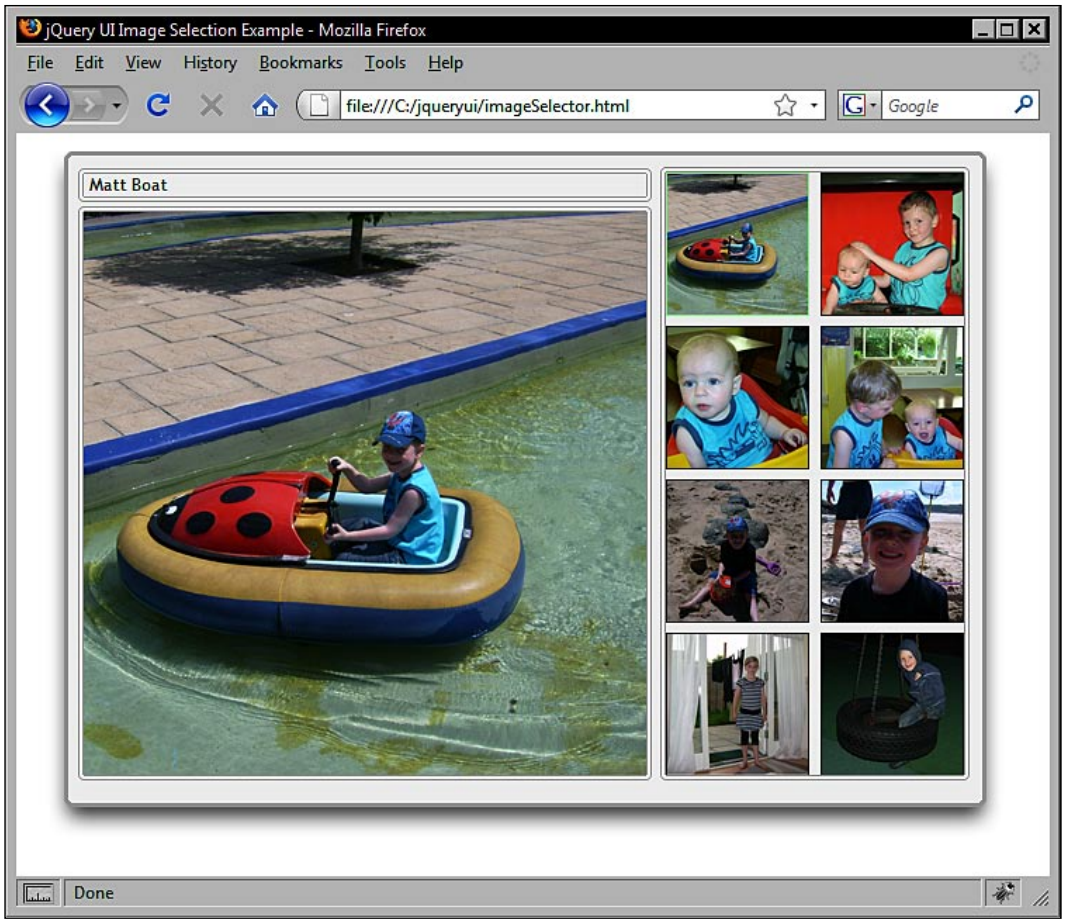



We could also very easily use the callback properties again to specify a function that changes the inner text of the disabled elements like we did in a previous example.

# Fun with selectables

In our final selectable example, we're going to make a basic image viewer. Images can be chosen for viewing by selecting the appropriate thumbnail.

Although this sounds like a relatively easy achievement, in addition to the actual mechanics of displaying the selected image, we'll also need to consider how to handle multiple selections. The following screenshot shows an example of what we'll end up with:




The images used in this example are provided in the code download because they need to be the correct size for this example to look right. There should be eight of both the large and thumbnail versions of each image, and the sizes of each are `100` by `100` pixels for the thumbnails and `400` by `400` pixels for the large versions.

We need to create two new folders called `large` and `thumbs` within our `img` directory. Then you should place the thumbnail images from the code download, or an equivalent number of equivalently sized images, in the `thumbs` folder and the full-sized images from the code download, or larger versions of your own thumbnails, into the `large` folder.

Let's get started with the code. In a fresh page in your text editor, add the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jquery.ui-1.5b4/
themes/flora/flora.tabs.css">
    <link rel="stylesheet" type="text/css" href="styles/
imageSelector.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Image Selection Example</title>
  </head>
  <body>
    <div id="imageSelector">
      <div id="status"></div>
      <div id="viewer"><span class="top"></span><span
class="bottom"></span></div>
      <div id="thumbs">
        <span class="top"></span>
        <img id="Matt_Boat" src="img/thumbs/a.jpg">
        <img id="Matt_and_James" src="img/thumbs/b.jpg">
        <img id="James" src="img/thumbs/c.jpg">
        <img id="Matt_and_James_2" src="img/thumbs/d.jpg">
        <img id="Matt_Beach" src="img/thumbs/e.jpg">
        <img id="Matt_Beach_2" src="img/thumbs/f.jpg">
        <img id="Beth" src="img/thumbs/g.jpg">
        <img id="Matt_Swing" src="img/thumbs/h.jpg">
        <span class="bottom"></span>
      </div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.selectable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        var x = 0;
```

```
        //define config object
        var selectOpts = {
          stop: function(e, ui) {
            ($(".ui-selected").length == 1) ? singleSelect() :
multiSelect();
          }
        };

        //make specified elements selectable
        $("#thumbs").selectable(selectOpts);

        function singleSelect() {

          //remove tabs if they already exist
          ($("#tabs").length != 0) ? $("#tabs").remove() : null;

          //add status bar if not present
          ($("#status").length == 0) ? $("<div>").attr("id",
"status").insertBefore($("#viewer")) : null;

          //add selected image to viewer
          $("#viewer").children().not("span").remove();
          $("<img>").attr("src", "img/large/" + $(".ui-selected")
.attr("src").substr($(".ui-selected").attr("src").length - 5,5))
.appendTo($("#viewer"));

          //clean file id and add to status bar
          $("#status").empty();
          var name = $(".ui-selected").attr("id");
          var matchIndex = name.indexOf("_");
          while(matchIndex != -1) {
            name = name.replace("_", " ")
            matchIndex = name.indexOf("_");
          }
          $("<p>").text(name).appendTo($("#status"));
        }

        function multiSelect() {

          //remove pre-existing clutter
          ($("#tabs").length != 0) ? $("#tabs").remove() : null;
          ($("#status").length != 0) ? $("#status").remove() : null;
          $("#viewer").children().not("span").remove();
          var x = 0;

          //create tab parent
          var tabList = $("<ul>").attr("id", "tabs").insertBefore
($("#viewer"));

          $(".ui-selected").each(function() {

              //add tabs
```

```
            var tabItem = $("<li>").appendTo(tabList);
            if ($(".ui-selected").length == 8) {
              var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:4}).appendTo(tabItem);
            } else if ($(".ui-selected").length == 7) {
              var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:6}).appendTo(tabItem);
            } else if ($(".ui-selected").length == 6) {
              var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:8}).appendTo(tabItem);
            } else if ($(".ui-selected").length == 5) {
              var tabLink = $("<a />").attr("href", "#" + x)
.css({paddingRight:10}).appendTo(tabItem);
            } else {
              var tabLink = $("<a />").attr("href", "#" +
x).appendTo(tabItem);
          }
          //clean file id and add span
          var name = $(this).attr("id");
          var matchIndex = name.indexOf("_");
          while(matchIndex != -1) {
            name = name.replace("_", " ")
            matchIndex = name.indexOf("_");
          }
         (name.indexOf("and") != -1) ? name = name.replace("and", "&")
: null;
         if ($(".ui-selected").length == 8) {
           $("<span />").text(name).css({paddingLeft:4})
.appendTo(tabLink);
         } else if ($(".ui-selected").length == 7) {
           $("<span />").text(name).css({paddingLeft:6})
.appendTo(tabLink);
         } else if ($(".ui-selected").length == 6) {
           $("<span />").text(name).css({paddingLeft:8})
.appendTo(tabLink);
         } else if ($(".ui-selected").length == 5) {
           $("<span />").text(name).css({paddingLeft:10})
.appendTo(tabLink);
         } else {
           $("<span />").text(name).appendTo(tabLink);
        }
         //add tab panels
         var panel = $("<div>").attr("id", x).appendTo($("#viewer"));
```

```
          $("<img>").attr("src", "img/large/" + $(this).attr("src")
.substr($(this).attr("src").length - 5,5)).appendTo(panel);

          x++;
        });

        //make the tab set and select first tab
        tabList.tabs();
    }
     });
   </script>
  </body>
</html>
```

Save this as `imageSelector.html`. It's a fairly large page so let's look at each part of it in turn. We'll start with the basic mark-up that the example is built on. We have a parent `<div>` with an `id` of `imageSelector` into which all of our other elements go.

Within the parent, we have a `<div>` that will act as a status bar and display the names of individually selected images, and a `<div>` which will act as the viewing panel and will display the full-sized version of the image.

Finally, we have our thumbnail images, which will be made selectable. The viewer and thumbs containers both have `<span>` elements nested inside them. These elements will be targeted by some CSS later to add the top and bottom borders of these two elements. Here is the code we just examined:

```
<div id="imageSelector">
  <div id="status"></div>
  <div id="viewer"><span class="top"></span><span class="bottom">
</span></div>
  <div id="thumbs">
    <span class="top"></span>
    <img id="Matt_Boat" src="img/thumbs/a.jpg">
    <img id="Matt_and_James" src="img/thumbs/b.jpg">
    <img id="James" src="img/thumbs/c.jpg">
    <img id="Matt_and_James_2" src="img/thumbs/d.jpg">
    <img id="Matt_Beach" src="img/thumbs/e.jpg">
    <img id="Matt_Beach_2" src="img/thumbs/f.jpg">
    <img id="Beth" src="img/thumbs/g.jpg">
    <img id="Matt_Swing" src="img/thumbs/h.jpg">
    <span class="bottom"></span>
  </div>
</div>
```

Following this mark-up, are the library files which are needed for this example and the final `<script>` block turning this into a working example. This is where the fun is. Again, we can look at what each part of the script does.

The first thing we do is create the `selectOpts` configuration object that our selectables will use. This object contains just one property which is the `stop` property. This property specifies a simple anonymous callback function. When this function is executed, it will either call the `singleSelect` or `multiSelect` function depending on the length of the jQuery object representing the selected selectables:

```
//function to execute when doc ready
$(function() {
  //define config object
  var selectOpts = {
    stop: function(e, ui) {
      ($(".ui-selected").length == 1) ? singleSelect() :
multiSelect();
    }
  };
```

We then initialize the selectables using our configuration object as an argument to the selectables constructor function:

```
  //make specified elements selectable
  $("#thumbs").selectable(selectOpts);
```

Following this, we can define our single and multiple selection handling functions. The `singleSelect` function begins by checking whether there is an element with an `id` of `tabs`. If there is, it removes it, and if there isn't, it does nothing. This is achieved using JavaScript's ternary expression:

```
  function singleSelect() {
    //remove tabs if they already exist
    ($("#tabs").length != 0) ? $("#tabs").remove() : null;
```

Next, the function checks whether there is a `status` element present. If it is not, then one is added so that the name of the image that has been selected can be displayed. If this element already exists (such as when the page initially loads), nothing is done:

```
    //add status bar if not present
    ($("#status").length == 0) ? $("<div>").attr("id", "status").inser
tBefore($("#viewer")) : null;
```

At this stage, we're almost ready to actually display the full-sized version of the image that has been selected. But before that is done, the next line of code clears out any residual elements from previous selections that are still in the viewer:

```
$("#viewer").children().not("span").remove();
```

Adding the full-sized image is extremely easy. First, we create a new image element, then we give it the `src` attribute that points to the large version of the thumbnail image by defining the path to the file as a string. We then add the file name extracted from the `src` attribute of the selected thumbnail:

```
    //add selected image to viewer
    $("<img>").attr("src", "img/large/" + $(".ui-selected").
attr("src").substr($(".ui-selected").attr("src").length - 5,5)).
appendTo($("#viewer"));
```

We could simply add the original image name to the status bar in its original form. However, the `id` of each image thumbnail has underscores in it, which looks untidy. It is simple enough to loop through the `id` of the selected thumbnail and replace any underscores with spaces. This 'clean' version of the name can then be added to a `<p>` element and inserted into the status bar. This brings us to the end of the `singleSelect` function:

```
    //clean file id and add to status bar
    $("#status").empty();
    var name = $(".ui-selected").attr("id");
    var matchIndex = name.indexOf("_");
    while(matchIndex != -1) {
      name = name.replace("_", " ")
      matchIndex = name.indexOf("_");
    }
    $("<p>").text(name).appendTo($("#status"));
  }
```

Next up is the `multiSelect` function, which is slightly larger but not much more complicated. We start off in the same way and check for the presence of tabs. If any are detected they will be removed.

We then check for the status element once again. This time, instead of creating it if it doesn't exist, we remove it if it *does* exist. We also empty the viewer as we did before. We initialize the `x` variable which will be used to give unique `id`s to the elements we are about to create:

```
function multiSelect() {
  //remove pre-existing clutter
  ($("#tabs").length != 0) ? $("#tabs").remove() : null;
```

```
($("#status").length != 0) ? $("#status").remove() : null;
$("#viewer").children().not("span").remove();
var x = 0;
```

To handle displaying multiple images in the viewer following a multiple selection, we will dynamically create a tab set. This allows us to use the same-sized element to display any number (well, eight anyway) of images.

First, we create and add the unordered list to the page that will hold the individual tabs. Then for each selected image, we create an `<li>` element and an `<a>` element. The creation of the link is rather convoluted and is necessary to make each tab thinner when there are more than four selected images. We basically loop through each possibility greater than four and give it the necessary padding:

```
    //create tab parent
    var tabList = $("<ul>").attr("id", "tabs").insertBefore($("#viewe
r"));
    $(".ui-selected").each(function() {
      //add tabs
      var tabItem = $("<li>").appendTo(tabList);
      if ($(".ui-selected").length == 8) {
        var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:4}).appendTo(tabItem);
      } else if ($(".ui-selected").length == 7) {
        var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:6}).appendTo(tabItem);
      } else if ($(".ui-selected").length == 6) {
        var tabLink = $("<a />").attr("href", "#" +
x).css({paddingRight:8}).appendTo(tabItem);
      } else if ($(".ui-selected").length == 5) {
        var tabLink = $("<a />").attr("href", "#" + x)
.css({paddingRight:10}).appendTo(tabItem);
      } else {
        var tabLink = $("<a />").attr("href", "#" +
x).appendTo(tabItem);
    }
```

Once we have created the list items and links, we clean the file `id`s as we did before. Additionally, we can replace any occurrences of the word **and** with an ampersand to save additional space in each tab:

```
    //clean file id and add span
    var name = $(this).attr("id");
    var matchIndex = name.indexOf("_");
    while(matchIndex != -1) {
```

```
        name = name.replace("_", " ")
        matchIndex = name.indexOf("_");
      }
      (name.indexOf("and") != -1) ? name = name.replace("and", "&") :
null;
```

We then use a similar conditional block to create a `<span>` element which will be added to each tab. Remember from the *Tabs* chapter that the `<span>` forms the label of the tab. The cleaned name is then added as the text of the `<span>`:

```
      if ($(".ui-selected").length == 8) {
$("<span />").text(name).css({paddingLeft:4}).appendTo(tabLink);
      } else if ($(".ui-selected").length == 7) {  $("<span />")
.text(name).css({paddingLeft:6}).appendTo(tabLink);
      } else if ($(".ui-selected").length == 6) {
$("<span />").text(name).css({paddingLeft:8}).appendTo(tabLink);
        } else if ($(".ui-selected").length == 5) {  $("<span />")
.text(name).css({paddingLeft:10}).appendTo(tabLink);
        } else {
          $("<span />").text(name).appendTo(tabLink);
        }
```

Once the required tabs have been created, we can then create the tab panels which will be used to hold the full-sized images. Each panel is given an `id` attribute using the `x` variable so that it matches the `href` of its tab heading. The tab panel is then added to the viewer and an image is created in the same way as before and then added to the panel. We also increment our `x` variable at this point:

```
        //add tab panels
        var panel = $("<div>").attr("id", x).appendTo($("#viewer"));
        $("<img>").attr("src", "img/large/" + $(this).attr("src")
.substr($(this).attr("src").length - 5,5)).appendTo(panel);

        x++;
      });
```

Finally, we use the tabs constructor method to turn our collection of list items and panels into a tab set:

```
      //make the tab set and select first tab
      tabList.tabs();  }
});
```

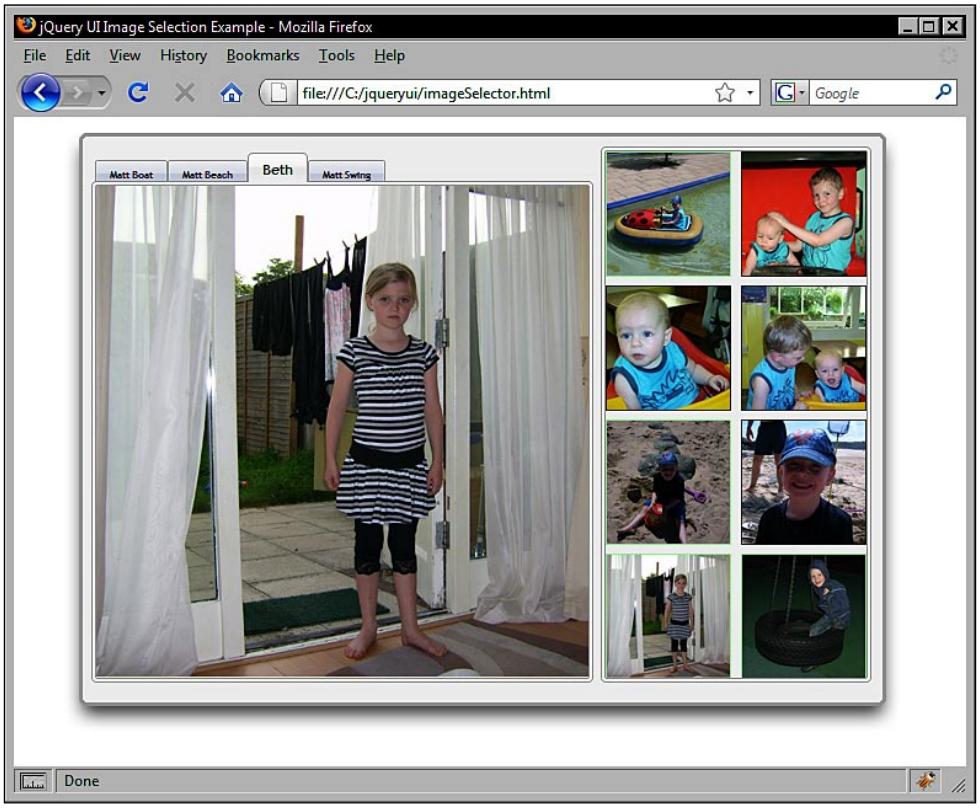
Save this as `imageSelector.html`. Our example is also heavily reliant on CSS to provide its overall appearance. In a new file in your text editor, create the following stylesheet:

```
#imageSelector {
  width:690px; height:500px; padding:5px;
  background:url(../img/image-selector/imageSelectorBG.gif) no-repeat;
  position:relative; margin:0 auto;
}
#status {
  width:408px; height:24px; position:absolute;
  top:17px; left:26px;
  background:url(../img/image-selector/imageStatus.gif) no-repeat;
  padding:3px 0 0 8px; font-family:"Trebuchet MS",Trebuchet,Verdana,
Helvetica,Arial,sans-serif;
  font-size:12px;
}
#viewer span, #thumbs span {
  height:4px; position:absolute; display:block;
}
#viewer {
  width:408px; height:408px;
  margin:-4px 10px 5px 0;
  background:url(../img/image-selector/imageTabsViewer.gif) repeat-y;
  position:absolute; left:26px; bottom:53px;
}
#viewer .top {
  width:408px;
  background:url(../img/image-selector/imageTabsTop.gif) no-repeat;
}
#viewer .bottom {
  width:408px;
  background:url(../img/image-selector/imageTabsBottom.gif) no-repeat;
  bottom:0;
}
#viewer img {
  position:absolute; left:4px; top:4px;
}
#thumbs {
  width:220px; height:428px;
  background:url(../img/imageThumbs.gif) repeat-y;
  position:absolute; top:20px; right:40px;
}
#thumbs img {
```

```
  border:1px solid #000; float:left;
  margin:0 4px 7px 4px; cursor:pointer;
}
#thumbs .top {
  width:220px;
  background:url(../img/image-selector/imageThumbsTob.gif) no-repeat;
  top:-4px; left:0px;
}
#thumbs .bottom {
  width:220px;
  background:url(../img/image-selector/imageThumbsBottom.gif)
no-repeat;
  bottom:-4px; left:0px;
}
#thumbs img.ui-selected { border:1px solid #99ff99; }

p { margin:0px; padding:0px; }
#tabs { position:absolute; top:25px; left:26px; }
.ui-tabs-panel {
  padding:0; border:0; background:transparent;
}
.ui-tabs-nav .ui-tabs-selected a {
  background-position:100% -26px;
}
.ui-tabs-nav .ui-tabs-selected a span {
  background-position:0% -26px; height:27px; font-size:11px;
}
.ui-tabs-nav a, .ui-tabs-nav a span {
background:url(../img/image-selector/imageTabsSprite.gif) no-repeat;
}
.ui-tabs-nav a span { height:25px; }
.ui-tabs-nav a:link, .ui-tabs-nav a:visited {
  color:#000; font-size:8px;
}
.ui-tabs-nav a {
  background-position:100% 0%; margin:2px 0 0 -2px;
}
.ui-tabs-nav li { position:relative; top:0px; }
.ui-tabs-nav li.ui-tabs-selected { top:-5px; }
```

Save this in the `styles` folder as `imageSelector.css`. We also need to create a new folder within our `img` folder to store some of the images used for this example. Create a new folder in `img` called `image-selector` and place the relevant images from the code download inside it.

When you run the example in a browser, you should see something like what is shown in the previous example. When multiple images have been selected, you should see tabs at the top of the viewer as in the following screenshot:




# Summary

The selectables component provides a powerful set of behaviors for related items. This enables us to easily provide users with a better means of selecting and manipulating sets of objects.

We first looked at the default implementation and then moved on to look at the two standard properties, along with the numerous callback properties, which can be used to perform different actions at different points in an interaction.

Finally, we looked at the methods exposed by this component's API. We saw that it had the usual range of methods for enabling, disabling, and removing functionality, and it also contains a `toggle` method, which reduces the amount of code by allowing us to do one of two things based on the current state of the component.

# 11
# Sorting

The final interaction helper that we're going to look at is the sortables component.

This component allows us to define one or more lists of elements (not necessarily actual `<ul>` or `<ol>` elements) where the individual items in the list(s) can be reordered.

The sortables component is like a specialized implementation of drag-and-drop, with a very specific role. It has an extensive API which caters for a wide range of behaviors and will be the focus of this chapter. We'll be looking at the following aspects of the component:

- A default sortable implementation
- The basic configurable properties
- The definition of a placeholder
- Sortable helpers
- Sortable items
- Connected Sortables
- Sortable's wide range of built-in event handlers
- A look at sortable's methods
- Submitting the sorted result to a server

## Basic implementation

A basic sortable list can be enabled with no additional configuration. Let's do this first so you can get an idea of the behavior enabled by this component. In a new file in your text editor, add the following code:

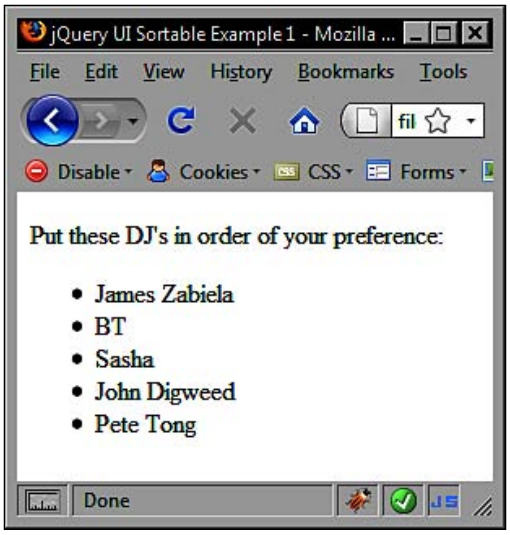
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
```

```
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 1</title>
  </head>
  <body>
    <p>Put these DJ's in order of your preference:</p>
    <ul id="sortables">
      <li>BT</li>
      <li>Sasha</li>
      <li>John Digweed</li>
      <li>Pete Tong</li>
      <li>James Zabiela</li>
    </ul>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //make specified element sortable
        $("#sortables").sortable();
      });
    </script>
  </body>
</html>
```

Save this as `sortable1.html`. On the page, we have a simple unordered list with five list items. There is no `flora` or `default` styling associated with this component so we don't need to link to any stylesheets in this basic example.

Code-wise, the default implementation is the same as it has been for each of the other components. We simply call the `sortable` constructor method on the parent `<ul>` element of the list items we want to make sortable.

Thanks to the sortables component, we should find that the individual list items can be dragged to different positions in the list, as in the following screenshot:




A lot of behaviors are added to the page. As we drag one of the list items up or down in the list, the other items automatically move out of the way creating a slot for the item that is currently being sorted to be dropped into. Additionally, when a sortable item is dropped, it will slide quickly but smoothly into its new position in the list. The library files that were needed for the basic implementation are as follows:

- `jquery-1.2.6.js`
- `ui.core.js`
- `ui.sortable.js`

As I mentioned earlier, the sortables component is a flexible addition to the library that can be applied to many different types of elements. For example, instead of using a list, we could use a series of `<div>` elements as the sortable list items:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 2</title>
  </head>
  <body>
    <div id="container">
```

```
        <p>Put these DJ's in order of your preference:</p>
        <div id="sortables">
          <div>BT</div>
          <div>Sasha</div>
          <div>John Digweed</div>
          <div>Pete Tong</div>
          <div>James Zabiela</div>
        </div>
      </div>
      <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
      <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
      <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
        //function to execute when doc ready
        $(function() {
          //make specified element sortable
          $("#sortables").sortable();
        });
      </script>
    </body>
  </html>
```

This can be saved as `sortable2.html`. As you can see, the behavior exhibited by this version is exactly the same as it was before. All that's changed is the underlying mark-up. Due to its simple base, we can also easily improve its appearance with some basic CSS. In a new file in your text editor, add the following code:

```
#container {
  width:272px; height:322px;
  background:url(../img/sortable_bg.gif) no-repeat;
  position:relative;
}
#container p {
  font-family:Arial; font-size:11px; position:absolute;
  width:100%; text-align:center; margin-top:20px;
}
#sortables { position:relative; top:45px; height:255px; }
#sortables div {
  height:35px; left:80px; position:relative; width:120px;
  padding-top:16px;
}
```

Save this in the `styles` folder as `sortable.css`. Link to the CSS file in `sortable2.html`, then save the change as `sortable3.html`. The underlying HTML and the JavaScript that drives it are identical, but with just a few CSS selectors and rules we can dramatically change the appearance of our example, as shown in the following screenshot:




# Configuring sortable properties

The sortables component has a huge range of configurable properties, many more than any of the other interaction components (but not as many as the date picker widget). The table below illustrates the range of properties at our disposal:

| Property | Default Value | Usage |
|---|---|---|
| `appendTo` | `parent` | Sets the element that helpers are appended to during a sort |
| `axis` | *none* | Constrains sortables to one axis of drag. Possible values are either `x` or `y` |
| `cancel` | `':input'` | Specifies elements that cannot be sorted |
| `connectWith` | `[]` | Specifies an array of separate lists of sortables so that sort items can be moved between each list |

| Property | Default Value | Usage |
|---|---|---|
| `containment` | `parent` | Constrains sortables to their container while they are being dragged. Values can be the strings `parent`, `window`, or `document`, or can be a jQuery selector |
| `cursor` | *none* | Defines the CSS cursor to apply while dragging a sortable |
| `delay` | `0` | Sets the time delay in milliseconds before the sort begins once a sortable item has been clicked (with the mouse button held down) |
| `distance` | `1` | Sets how far in pixels the mouse pointer should move while the left button is held down before the sort should begin |
| `dropOnEmpty` | `true` | Allows linked items to be dropped onto empty slots |
| `forcePlaceholderSize` | `false` | Forces the placeholder to have a size. The placeholder is the empty space that a sortable can be dropped on to |
| `grid` | `[]` | Sets sortables to snap to a grid while being dragged. Value should be an array with 2 items; the x and y distances between gridlines |
| `handle` | *none* | Specifies an element to be used as the drag handle on sortable items |
| `helper` | `original` | Specifies a helper element that will be used as a proxy element while the sortable is being dragged. Can accept a function that returns an element |
| `items` | `'>*'` | Specifies the items that should be made sortable. The default makes all children sortable |
| `opacity` | `1` | Specifies the CSS opacity of the element being sorted |
| `placeholder` | *none* | Specifies a CSS class to be added to empty slots |
| `revert` | `true` | Enables animation when moving sortables into their new slots |
| `scroll` | `true` | Enables page scrolling when a sortable is moved to the edge of the viewport |

| Property | Default Value | Usage |
|---|---|---|
| scrollSensitivity | 20 | Sets how close a sortable must get, in pixels, to the edge of the viewport before scrolling should begin |
| scrollSpeed | 20 | Sets the distance in pixels that the viewport should scroll when a sortable is dragged within the sensitivity range |
| zIndex | 1000 | The CSS z-index of the sortable/helper while being dragged |

Let's work some of these properties into our previous example to get a feel for the effect they have on the behavior of the component. Change `sortable3.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/sortable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 4</title>
  </head>
  <body>
    <div id="container">
      <p>Put these DJ's in order of your preference:</p>
      <div id="sortables">
        <div>BT</div>
        <div>Sasha</div>
        <div>John Digweed</div>
        <div>Pete Tong</div>
        <div>James Zabiela</div>
      </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var sortOpts = {
```

```
            axis: "y",
            containment: "#container",
            cursor: "move",
            distance: 30
          };

          //make specified element sortable
          $("#sortables").sortable(sortOpts);
        });
      </script>
    </body>
  </html>
```

Save this as `sortable4.html`. We use four properties in our configuration object; the `axis` property, the value of which we have specified as `y` to restrain the motion of the sortable currently being dragged to just up and down.

We use the `containment` property, specifying a jQuery selector for the element that the sortables should be contained within. Care should be taken with this property; if we had specified `#sortables` as the container, we would have not been able to move items into the top or bottom positions.

We also specify the `cursor` property which automatically adds the CSS `move` icon. Like the draggable, the CSS move icon is not actually displayed until the sort begins.

Finally, we configure the `distance` property with a value of `30` which specifies that the mouse pointer should move 30 pixels before the sort begins. The `distance` property works in the same way with sortables as it did with draggables earlier in the book and is great for preventing unwanted sorts, but in practice we'd probably use a much lower threshold than 30 pixels.

The effects of these properties can easily be seen when the page is run in a browser:




Let's look at some more properties. Change `sortable4.html` so that it appears as follows instead:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableHandle.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 5</title>
  </head>
  <body>
    <div id="container">
      <p>Put these DJ's in order of your preference:</p>
      <div id="sortables">
        <div>BT<div class="handle"></div></div>
        <div>Sasha<div class="handle"></div></div>
        <div>John Digweed<div class="handle"></div></div>
        <div>Pete Tong<div class="handle"></div></div>
```

```
          <div>James Zabiela<div class="handle"></div></div>
        </div>
      </div>
      <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
      <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
      <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
      <script type="text/javascript">
        //function to execute when doc ready
        $(function() {
          //define config object
          var sortOpts = {
            revert: "slow",
            handle: ".handle",
            delay: 1000,
            opacity: 0.5
          };
          //make specified element sortable
          $("#sortables").sortable(sortOpts);
        });
      </script>
    </body>
</html>
```

Save this as `sortable5.html`. The `revert` property has a default value of `true`, but can also take one of the speed string values (`slow`, `normal`, or `fast`) that we've seen in other animation properties.

The `delay` property accepts a value in milliseconds that the component should wait before allowing the sort to begin. This property won't prevent the sort from occurring, even if the mouse button is let go of, or the pointer is moved away from the sortable. It will still get 'picked up' after the specified time has elapsed.

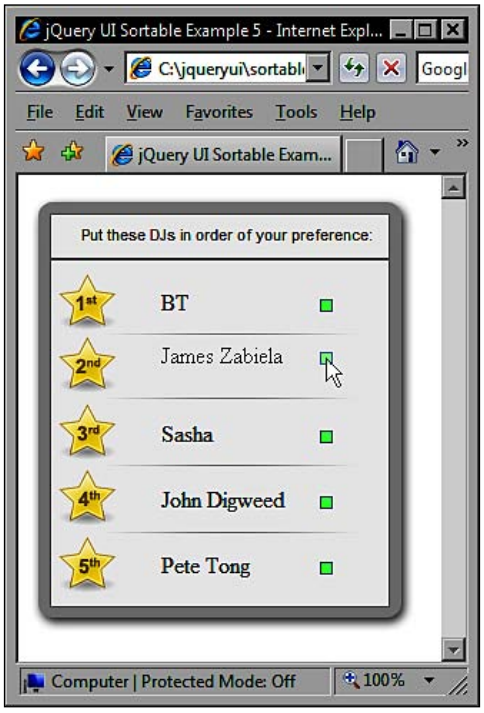
The value of the `opacity` property is used to specify the CSS opacity of the element that is being sorted while the sort takes place. The value should be a floating-point number between `0` and `1`, with `1` corresponding to no opacity and `0` specifying full opacity. Note that the `opacity` property can affect the way that IE renders text.

One of the properties we've used is the `handle` property which allows us to define a region within the sortable which must be used to initiate the sort. Dragging on other parts of the sortable will not cause the sortable to be dragged.

The handles have been styled with some CSS, so we'll need to update `sortable.css` as well. There is no need to look at the whole file again. Just add the following new selector and rules to the end of the file:

```
#sortables div.handle {
  border:1px solid #003399; position:absolute; top:20px;
  margin-left:20px; width:7px; height:7px; background-color:#66FF66;
}
```

Save the changes as `sortableHandle.css`. You can see how the handle will appear in the following screenshot:




# Placeholders

A placeholder defines the empty space, or slot, that is left while one of the sortables is *en sort* to its new position. The placeholder isn't rigidly positioned, it will dynamically move to whichever sortable has been displaced by the movement of the sortable that is being sorted.

There are two properties that are specifically concerned with placeholders; the very aptly named `placeholder` property and the `forcePlaceholderSize` property.

The `placeholder` property allows you to define a CSS class that should be added to the placeholder while it is empty. This is a useful property that we can use often in our implementations.

The `forcePlaceholderSize` property, set to `false` by default, is a property that we'll probably use less often. The placeholder will automatically assume the size of the sortable item, which in most cases is fine.

In a new file in your text editor, add the following code:

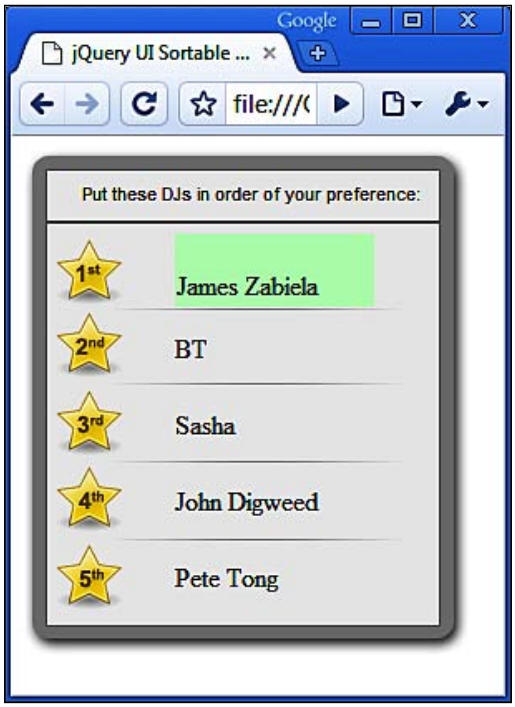
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortablePlaceholder.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 6</title>
  </head>
  <body>
    <div id="container">
      <p>Put these DJ's in order of your preference:</p>
      <div id="sortables">
        <div>BT</div>
        <div>Sasha</div>
        <div>John Digweed</div>
        <div>Pete Tong</div>
        <div>James Zabiela</div>
      </div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var sortOpts = {
          placeholder: "empty"
        };
```

```
        //make specified element sortable
        $("#sortables").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this as `sortable6.html`. We've specified the name of the class that we want to add to the placeholder. Remember this is a class name not a class selector, so no period is used at the start of the string. Next, we should add the selector and rules to our CSS file. The CSS file we use is exactly the same as our base CSS file (not the one from the previous example) with the following code added to the end:

```
.empty { background-color:#cdfdcd; }
```

Save this as `sortablePlaceholder.css` in the `styles` folder. When we run the new HTML file in a browser, we should be able to see the specified styles applied to the placeholder while the sort is taking place:



# Sortable helpers

We looked at helper/proxy elements back when we looked at the draggables component in the last chapter. Helpers can also be defined for sortables which function in a similar way to those of the draggable component, although there are some subtle differences in this implementation.

With sortables, the original sortable is hidden when the sort interaction begins and a clone of the original element is dragged instead. So with sortables, helpers are an inherent feature.

Like with draggables, the `helper` property of sortables may take a function as its value. The function, when used, will automatically receive the event object and original sortable element as arguments and should return the element to use as the helper. Although it's very similar to the draggable helper example, let's take a quick look at it when used in conjunction with sortables. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/sortable.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 7</title>
  </head>
  <body>
    <div id="container">
      <p>Put these DJ's in order of your preference:</p>
      <div id="sortables">
        <div>BT</div>
        <div>Sasha</div>
        <div>John Digweed</div>
        <div>Pete Tong</div>
        <div>James Zabiela</div>
      </div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
```

```
        var sortOpts = {
          helper: helperMaker
        };
        //define function that returns helper element
          function helperMaker(e, ui) {
                  return $("<div>").css({
                     border:"4px solid #cccccc",
                     opacity:"0.5"
                   });
        }

        //make specified element sortable
        $("#sortables").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this file as `sortable7.html`. We have our `helperMaker` function which creates and returns the element that is to be used as the helper while the sort is in progress. We can set some basic CSS properties on the new element so that we don't need to provide additional rules in the stylesheet.

The following screenshot shows how the helper will appear while in motion:



# Sortable items

By default, all children of the element that the `sortable` method is called on are turned into sortables (except those specified in the `cancel` property). While this is a useful feature of the component, there may be times when we don't necessarily want all child elements to become sortable.

The `items` property controls which child elements of the specified element should be made sortable. It makes all child elements sortable using `>*` as its default value, but we can alter this to only specify the elements we want. In a new file in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableItems.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 8</title>
  </head>
  <body>
    <div id="container">
      <p>Put these DJ's in order of your preference:</p>
      <div id="sortables">
        <div class="sortee">BT</div>
        <div class="sortee">Sasha</div>
        <div class="sortee">John Digweed</div>
        <div class="sortee">Pete Tong</div>
        <div class=" unsortable ">James Zabiela</div>
      </div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var sortOpts = {
          items: ".sortee"
```

```
        };
        //make specified element sortable
        $("#sortables").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

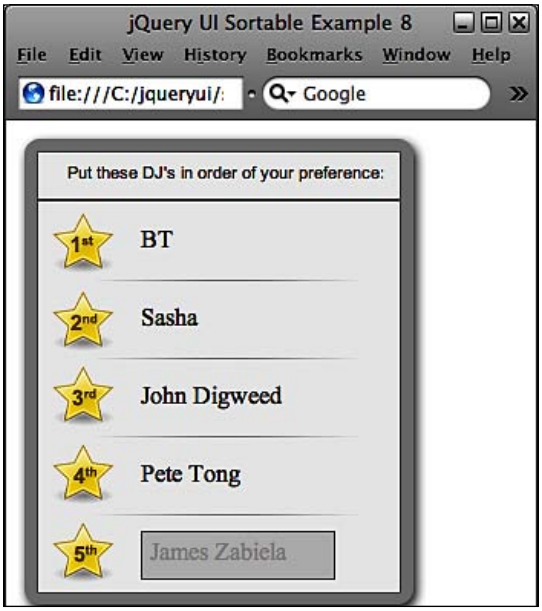
Save this as `sortable8.html`. We've added a class name of `sortee` to the most of the original `<div>` elements within our sortable container, and have also added the class name `unsortable` to the last item.

In our `<script>`, we've specified `sortee` as the value of the `items` property, so all of our `<div>` elements with the class name `sortee` will be sortable, while the `<div>` with the class name `unsortable` will not.

The new CSS used to style the `unsortable` element can be as simple as the following selector and rules, which should be added to `sortable.css`:

```
#sortables div.unsortable {
  border:1px solid #000; background-color:#CCCCCC;
  height:26px; padding:4px 0 0 5px; top:11px; color:#adabab;
}
```

Save this as `sortableItems.css` in the `styles` folder. Try the new page out, the following screenshot shows what you should see:



# Connected lists

So far, the examples that we have looked at have all centered around a single list of sortable items. What happens when we want to have two lists of sortable items, and more importantly, can we move items from one list to another?

Having two sortable lists is of course extremely easy and involves simply defining two containers and their child elements, and then independently passing each container to the `sortable` constructor method.

Allowing separate lists of sortables to exchange and share sortables is also extremely easy thanks to the `connectWith` property, which allows us to define an array of sortable containers whose sortables can move between them. Let's look at this in action. In a new file in your text editor, add the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableConnected.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 9</title>
  </head>
  <body>
    <p>Tell us what music you like and don't like:</p>
    <div id="likes">
      <p>Likes</p>
      <div>House</div>
      <div>Hip Hop</div>
      <div>Breaks</div>
      <div>Drum & Bass</div>
      <div>Rock</div>
    </div>
    <div id="dislikes">
      <p>Dislikes</p>
      <div>Folk</div>
      <div>Country</div>
      <div>Pop</div>
      <div>Classical</div>
      <div>Opera</div>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var sortOpts = {
          items: "div",
          connectWith: ["#likes", "#dislikes"]
        };

        //make specified elements sortable
        $("#likes").sortable(sortOpts);
        $("#dislikes").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this as `sortable9.html`. Everything on the page is pretty similar to what we have worked with before. There are just simple collections of nested `<div>` elements with some explanatory text. Within our final `<script>` tag however, we have some unfamiliar, although still very simple, code.

We still define a single configuration object which is shared between both sortable elements. We're using the `items` property once again to ensure that the `<p>` elements that form the box headings within our sortable containers aren't sortable themselves.

The `connectWith` property takes an array containing the jQuery `id` selectors for both of the sortable containers and it is this that allows us to share individual sortables between the two elements.

We saw a moment ago that both sortable elements share the same configuration object, this is important to understanding how the `connectWith` property functions. This property only provides a one-way transmission of sortables, so if we were to only use the configuration object in the `likes` sortable and specify just the `id` of the `dislikes` sortable, we would only be able to move items from `likes` to `dislikes`, not the other way.

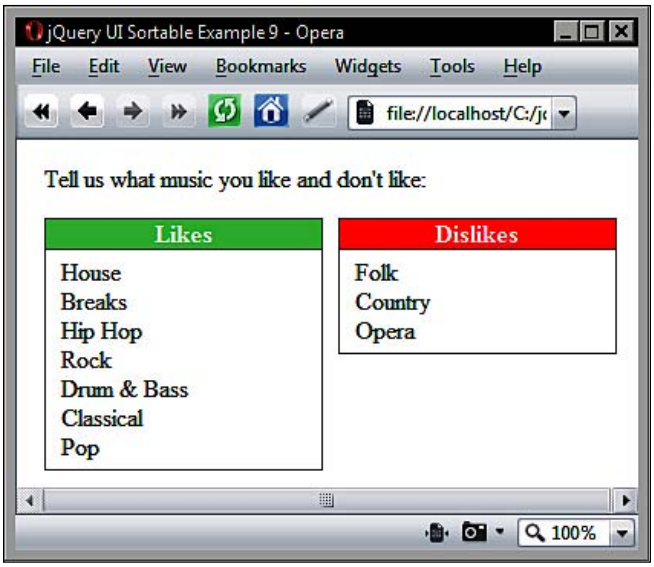
Specifying both sortables' `ids` and using the configuration objects in both constructor functions allows us to move items between both elements, and allows us to cut down on coding. We could also use the following `<script>` tag (although it would be less efficient):

```
<script type="text/javascript">
  //function to execute when doc ready
  $(function() {

    //define config objects
    var sortOpts = {
      items: "div",
      connectWith: ["#dislikes"]
    };
    var sortOpts2 = {
      items: "div",
      connectWith: ["#likes"]
    };

    //make specified elements sortable
    $("#likes").sortable(sortOpts);
    $("#dislikes").sortable(sortOpts2);
  });
</script>
```

This code will work completely as is with no styling whatsoever, but for aesthetic purposes, we may use any arbitrary CSS to make things look as we wish. The following CSS for example is more than adequate in giving an impression of how the page could look:

```
p { position:relative; left:10px; }
#likes, #dislikes {
  width:180px;
  border:1px solid #000;
  float:left;
  margin-left:10px; padding-bottom:5px;
}
#likes p, #dislikes p {
  margin:0px 0 5px;
  text-align:center; font-weight:bold;
  border-bottom:1px solid #000;
  color:#fff;
  left:0px;
}
#likes p { background-color:#66CC66; }
#dislikes p { background-color:#FF0000; }
#likes div, #dislikes div { margin-left:10px; }
```

Save this as `sortableConnected.css` in your `styles` folder. When you run the page in your browser, you should find that not only can the individual items be sorted in their respective elements, but that items can also be moved between elements, as shown in the following screenshot:




# Reacting to sortable events

In addition to the already large list of configurable properties defined in the sortables class, there are a whole load more in the form of callback properties which can be passed functions to execute at different points during a sortable interaction. These are listed in the following table:

| Callback | Fired |
|---|---|
| `activate` | When sorting starts on a connected list |
| `beforeStop` | When the sort has stopped but the original slot is still available |
| `change` | During a sort, when the DOM position of the sortable has changed |
| `deactivate` | When sorting stops on a connected list |
| `out` | When a sortable is moved away from a connected list |
| `over` | When a sortable is over a connected list |
| `receive` | When a sortable is received from a connected list |
| `remove` | When a sortable is moved from a connected list |

| Callback | Fired |
|---|---|
| `sort` | When a sort is taking place |
| `start` | When the sort starts |
| `stop` | When the sort ends |
| `update` | When the sort has ended and the DOM position has changed |

Event handlers such as these are important because they allow us as the programmers to react to specific things occurring. Each of the components that we've looked at in the preceding chapters have defined their own suite of custom events and the sortables component is certainly no exception.

Many of these events will fire during any single sort interaction. The following list shows the order in which they will fire:

- `start`
- `sort`
- `change`
- `beforeStop`
- `stop`
- `update`

As soon as one of the sortables is 'picked up', the `start` event is triggered. Following this, on *every single mouse move* the `sort` event will fire, making this event very intensive. As soon as another item is displaced by the current sortable, the `change` event is fired. Once the sortable is 'dropped', the `beforeStop` and `stop` events fire and if the sortable is now at a different position, the `update` event is fired last of all.

For the next few examples, we'll work some of these event handling properties into the previous example, starting with the `start` and `stop` events. Change `sortable9.html` so that it appears as follows (new code is shown in bold):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableConnected.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 10</title>
  </head>
  <body>
    <p>Tell us what music you like and don't like:</p>
```

```
    <div id="likes">
      <p>Likes</p>
      <div>House</div>
      <div>Hip Hop</div>
      <div>Breaks</div>
      <div>Drum & Bass</div>
      <div>Rock</div>
    </div>
    <div id="dislikes">
      <p>Dislikes</p>
      <div>Folk</div>
      <div>Country</div>
      <div>Pop</div>
      <div>Classical</div>
      <div>Opera</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var sortOpts = {
          items: "div",
          connectWith: ["#likes", "#dislikes"],
          start: function(e, ui) {
            $("<p>").text("The active sort item is " + ui.helper.
text()).css({clear:"both"}).attr("id", "message").appendTo("body");
          },
          stop: function() {
            $("#message").remove();
          }
        };
        //make specified elements sortable
        $("#likes").sortable(sortOpts);
        $("#dislikes").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this as `sortable10.html`. Our event usage in this example is minimal. When the sort starts, we simply create a new paragraph element and add some text to it, including the text content of the element that is being sorted. The text message is then duly appended to the `<body>` of the page. When the sort stops, we simply remove the text.

Using the second object passed to the callback function is very easy as you can see. The object itself refers to the parent sortables container, and the `helper` property refers to the actual item being sorted (or its helper). As this is a jQuery object, we can call jQuery methods, like `text`, on it.

When you run the page, the message should appear briefly until the sort ends, at which point it's removed:




Let's look at a couple more of these simple callbacks before we move on to look at the additional callbacks used with connected sortables. Change `sortable10.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableConnected.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
```

```
    <title>jQuery UI Sortable Example 11</title>
  </head>
  <body>
    <p>Tell us what music you like and don't like:</p>
    <div id="likes">
      <p>Likes</p>
      <div>House</div>
      <div>Hip Hop</div>
      <div>Breaks</div>
      <div>Drum & Bass</div>
      <div>Rock</div>
    </div>
    <div id="dislikes">
      <p>Dislikes</p>
      <div>Folk</div>
      <div>Country</div>
      <div>Pop</div>
      <div>Classical</div>
      <div>Opera</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define object to hold start and end lists
        var list = {
          start: "",
          end: ""
        };
        //define config object
        var sortOpts = {
          items: "div",
          connectWith: ["#likes", "#dislikes"],
          start: function(e, ui) {
            list.start = ui.helper.parent().attr("id");
          },
          change: function(e, ui) {
            ($("#message")) ? $("#message").remove() : null;
            var pos = ui.absolutePosition.top;
```

```
            (pos < 90) ? pos = "First" : null;
            (pos < 110) ? pos = "Second" : null;
            (pos < 130) ? pos = "Third" : null;
            (pos < 150) ? pos = "Fourth" : null;
            (pos < 170) ? pos = "Fifth" : null;
            (pos < 190) ? pos = "Sixth" : null;
            $("<p>").text(ui.helper.text() + " is now at " + pos + "
place").css({clear:"both"}).attr("id", "message").appendTo("body");
          },
          update: function(e, ui) {
            list.end = $(this).attr("id");
            ($("#message")) ? $("#message").remove() : null;
            $("<p>").text(ui.helper.text() + " started in " + list.
start + " and now belongs to " + list.end).css({clear:"both"}).
attr("id", "message").appendTo("body");
          }
        };

        //make specified elements sortable
        $("#likes").sortable(sortOpts);
        $("#dislikes").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this as `sortable11.html`. In this example, we work with the `start`, `change`, and `update` callbacks and also with the `ui.absolutePosition.top` and `ui.helper` properties.

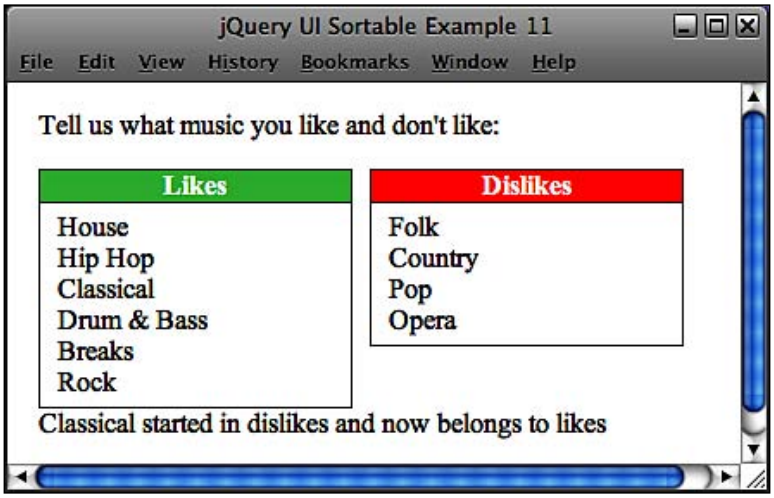
We first create an object that will be used to store the `id`s of the list that the sortable begins in and the list that it ends up in. The values for the properties in the object are set later in the `<script>`.

Our first anonymous callback function, triggered when the sort begins, simply gets the `id` of the sort element's parent, which will be the list that it starts in. This is written to our `list` object as the value of the `start` property.

The next function, triggered every time the current sort item pushes another item out of its placeholder, determines how far the current sort item is from the top of the page using the `top` property of the `ui.absolutePosition` object, and uses this to add a message to the page indicating its new 'rank' in the list.

At the end of the sort interaction (provided the sort item has displaced at least one other item), the `update` callback first gets the `id` of the current list using `$(this)`, and sets this as the value of the `end` property in the `list` object. Finally, it adds a message indicating the list that the item started in and the list that it ended up in using the properties that we set in our `list` object. Note that the displayed message will not work correctly sometimes for lists with more than five items.

The following screenshot shows how the page should look following a sort interaction:




# Connected callbacks

Six of the available callback properties exposed by sortables can be used in conjunction with connected sortables. These events fire at different times during an interaction alongside the events that we have already looked at.

Like the standard unconnected events, not all of the connected events will fire in any interaction. Some events, like `over`, `off`, `remove`, and `receive` for example, will only fire if a sort item moves to a new list. Other events, such as the `activate` and `deactivate` events, will fire in all executions, whether any sort items change lists or not. Additionally, some connected events, such as `activate` and `deactivate`, will fire for each connected list.

Provided at least one item is moved between lists, events will fire in the following order:

- `start`
- `activate`
- `sort`
- `change`
- `beforeStop`
- `stop`
- `remove`
- `update`
- `receive`
- `deactivate`

Let's now see some of these connected events in action. Change `sortable11.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableConnected.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 12</title>
  </head>
  <body>
    <p>Tell us what music you like and don't like:</p>
    <div id="likes">
      <p>Likes</p>
      <div>House</div>
      <div>Hip Hop</div>
      <div>Breaks</div>
      <div>Drum & Bass</div>
      <div>Rock</div>
    </div>
    <div id="dislikes">
      <p>Dislikes</p>
      <div>Folk</div>
      <div>Country</div>
      <div>Pop</div>
      <div>Classical</div>
```

```
      <div>Opera</div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //define config object
        var sortOpts = {
          items: "div",
          connectWith: ["#likes", "#dislikes"],
          activate: function() {
            $("<p>").text($(this).attr("id") + " has been activated")
.css({clear:"both"}).attr("id", "message").appendTo("body");
          },
          deactivate: function() {
            $("<p>").text($(this).attr("id") + " has been
deactivated").css({clear:"both"}).attr("id", "message")
.appendTo("body");
          },
          receive: function(e, ui) {
            $("<p>").text(ui.helper.text() + " has joined a new
list").css({clear:"both"}).attr("id", "message").appendTo("body");
          },
          remove: function(e, ui) {
            $("<p>").text(ui.helper.text() + " has left its original
list").css({clear:"both"}).attr("id", "message").appendTo("body");
          }
        };
        //make specified elements sortable
        $("#likes").sortable(sortOpts);
        $("#dislikes").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this as `sortable12.html`. The `activate` and `deactivate` events are fired for each connected list at the start of any sort interaction. As these events are executed in the context of each sortable, we can use `$(this)` to refer to each sortable instead of using the second object that is automatically passed to each of our functions.

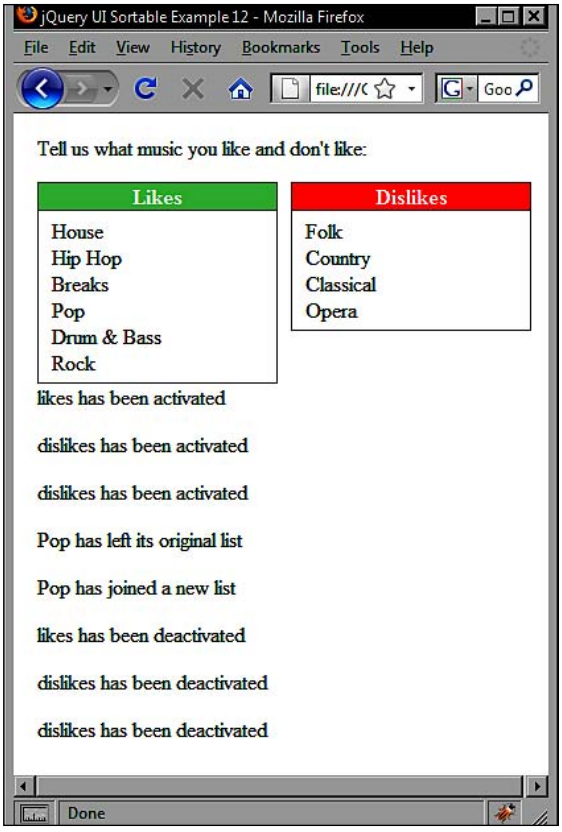
The `remove` and `receive` events are fired once each time a sort item moves from one list to another, and as with previous examples, we can easily make use of the objects passed to our functions.

When we run this example in a browser however, we notice that something unusual is happening. Everything works as it should, events fire and the information that we want to get is readily available. But, our `activate` and `deactivate` methods each seem to be firing an additional time.

This behavior can be 'fixed' by providing a separate configuration object for each of the sortable constructor functions. When we do this, the events fire once only for each list as is expected.

This unexpected behavior is not necessarily a problem however, because if you notice, the additional event that is fired changes depending on which of the sortables is interacted with. So it can be used to easily refer to the original sortable.

The following screenshot shows how the page should appear when an item is moved between sortables:



# Sortable methods

The sortables component exposes the usual set of methods for making the component 'do things', and like the selectables component that we looked at before, it also defines a couple of unique methods not seen in any of the other components. The following table lists sortable's full range of methods:

| Method | Use |
|---|---|
| `destroy` | Completely removes sortable functionality |
| `disable` | Temporarily removes sortable functionality |
| `enable` | Restores sortable functionality |
| `refresh` | Triggers the reloading of the set of sortables |
| `refreshPositions` | Triggers the cached refresh of the set of sortables |
| `serialize` | Constructs a URL-appendable string for sending new sort order to the server |
| `toArray` | Serializes the sortables into an array of strings |

Most of these methods we've seen before in various forms under the other components that we have used. Methods that we have not seen before however are `refreshPositions`, `serialize`, and `toArray`.

The `refreshPositions` method is similar to the `refresh` method except that it refreshes the cached positions of the sortables. This method is called automatically by the component at the appropriate time, but is also available for us to make use of if need be, although its use should be limited where possible because of its intensity.

The `serialize` method is an important one for doing something useful with the resulting post sort sortables. It's specially formulated for turning the on-page elements into a simple string format that is easy to pass across the network to a waiting server-side application. Let's see this in action. In a new file in your text editor, create the following page:

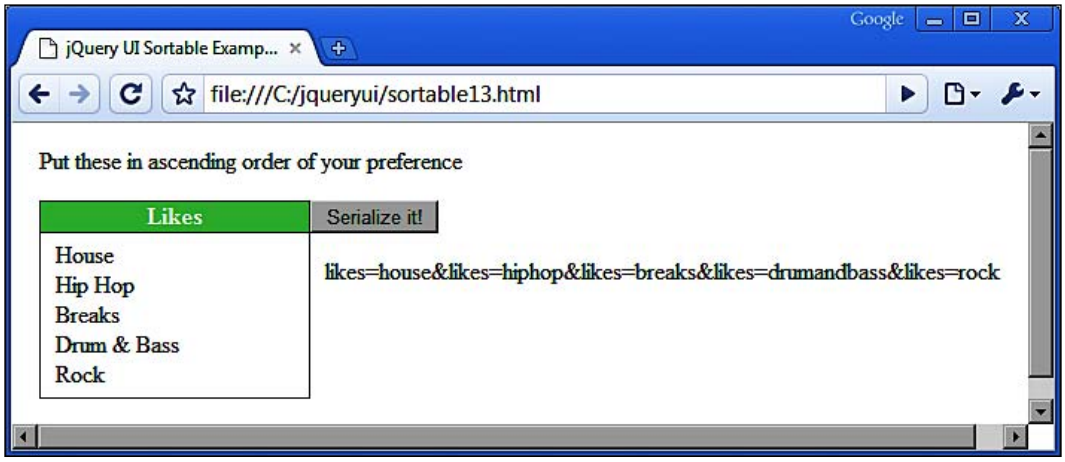
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
sortableConnected.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Example 13</title>
  </head>
  <body>
```

```
    <p>Put these in ascending order of your preference</p>
    <div id="likes">
      <p>Likes</p>
      <div id="likes_house">House</div>
      <div id="likes_hiphop">Hip Hop</div>
      <div id="likes_breaks">Breaks</div>
      <div id="likes_drumandbass">Drum & Bass</div>
      <div id="likes_rock">Rock</div>
    </div>
    <button id="serialize">Serialize it!</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //define config object
        var sortOpts = {
          items: "div",
        };

        //make specified element sortable
        $("#likes").sortable(sortOpts);

        //handler for button click
        $("#serialize").click(function() {
          var serialized = $("#likes").sortable("serialize", {
key:"likes"});
          (!$("#string")) ? null : $("#string").remove();
          $("<p>").attr("id", "string").text(serialized).
appendTo("body");
        });
      });
    </script>
  </body>
</html>
```

Save this as `sortable13.html`. We've dropped the second set of sortables for this example and added a button to the page which triggers the serialization. We've also added `id` attributes to each of the sortable items in the format of the name of the parent sortable (likes) and the individual items separated by an underscore.

The click handling function simply serializes the sortable elements by calling the `serialize` method and then checks for the presence of a previous message on the page. If the message exists (if the button has already been clicked), it is removed and the serialized string is added to the page for us to see. We use the `key` configuration property of the `serialize` method to set the list id as the first part of each item in the serialized string.

The following screenshot shows what you should see when you run the page in your browser and click the **Serialize it!** button (and, optionally, perform an actual sort):




As you can see, the format of the serialized string is quite straight-forward. The sortable items appear in the order that the items appear on the page and are separated by an ampersand. Each serialized item is made up of two parts; the name of the sortable to which they belong and the individual item, separated (by default, but can be changed) by the = character.

If serialization is a term you've never come across before, and as no native serialization methods exist within JavaScript, this would be no surprise. Don't worry as you've probably used it before (or at least its opposite deserialization) without even realizing.

When data is converted into JSON so that you can download it and process it directly in the browser, it is serialized into a format suitable for transportation across the Internet. When you process the JSON object on the client-side to extract the data within it, you are in effect deserializing, or parsing it.

You might be wondering why the method doesn't serialize the sortable into a JSON object to pass back to the server. The main reason is because the output of the serialize method is in the format that backend code, such as PHP will automatically be expecting.

In the previous example, all we do is display the serialized string on the page, but the string is in the perfect format for use with jQuery's `ajax` method, or to appending to a URL, to pass the resulting string to a server for further processing.

The component uses a regular expression to read the `id`s of each sortable item and split them into the set name and item name format found in the outputted string. It is possible to supply an alternative expression using a literal configuration object passed to the `serialize` method. It is also possible to use an alternative attribute than `id` to build the serialized string.

The properties available for use with this method are listed in the following table:

| Property | Default Value | Usage |
|---|---|---|
| `attribute` | `id` | Specifies the `id` to use as the item name in the parsed string |
| `connected` | `false` | If set to `true` serialization, will include all connected lists |
| `expression` | `"(.+)[-=_](.+)"` | The expression used to parse the specified attribute of each sortable item |
| `key` | The first result of expression | Specifies the key to be used as the property of each item in the serialized output |

The `toArray` method works in a similar way to serialize, except that with `toArray`, the output is not a string but an array of strings. This gives us an object that can easily be passed to other widgets.

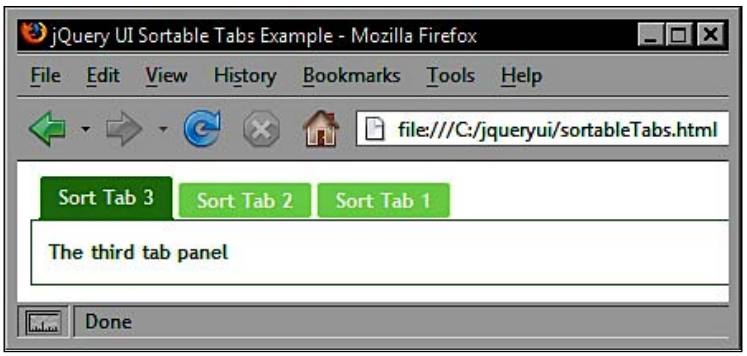
# Widget compatibility

In the previous chapter, we saw that both the resizables and the selectables component worked well with the tabs widget (and we already know how well the dialog and resizables components go together). The sortable component is also highly compatible with other widgets. Let's look at a basic example. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="jqueryui1.6rc2/
themes/flora/flora.tabs.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Sortable Tabs Example</title>
  </head>
  <body>
    <ul id="tabs">
      <li><a href="#0"><span>Sort Tab 1</span></a></li>
      <li><a href="#1"><span>Sort Tab 2</span></a></li>
      <li><a href="#2"><span>Sort Tab 3</span></a></li>
    </ul>
    <div id="0">The first tab panel</div>
    <div id="1">The second tab panel</div>
    <div id="2">The third tab panel</div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.tabs.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        $("#tabs").tabs();

        //define config object
        var sortOpts = {
          axis: "x",
          items: "li"
        };

        //make specified elements sortable
        $("#tabs").sortable(sortOpts);
      });
    </script>
  </body>
</html>
```

Save this page as `sortableTabs.html`. There is nothing in the code that we haven't seen before so we won't go into any great detail about it. When you run the page in your browser, you should see that the components work in exactly the way that we want them to. The tabs can be sorted horizontally to any order, but as the tabs are linked to their panel by `href`, they will still refer to the correct panel.

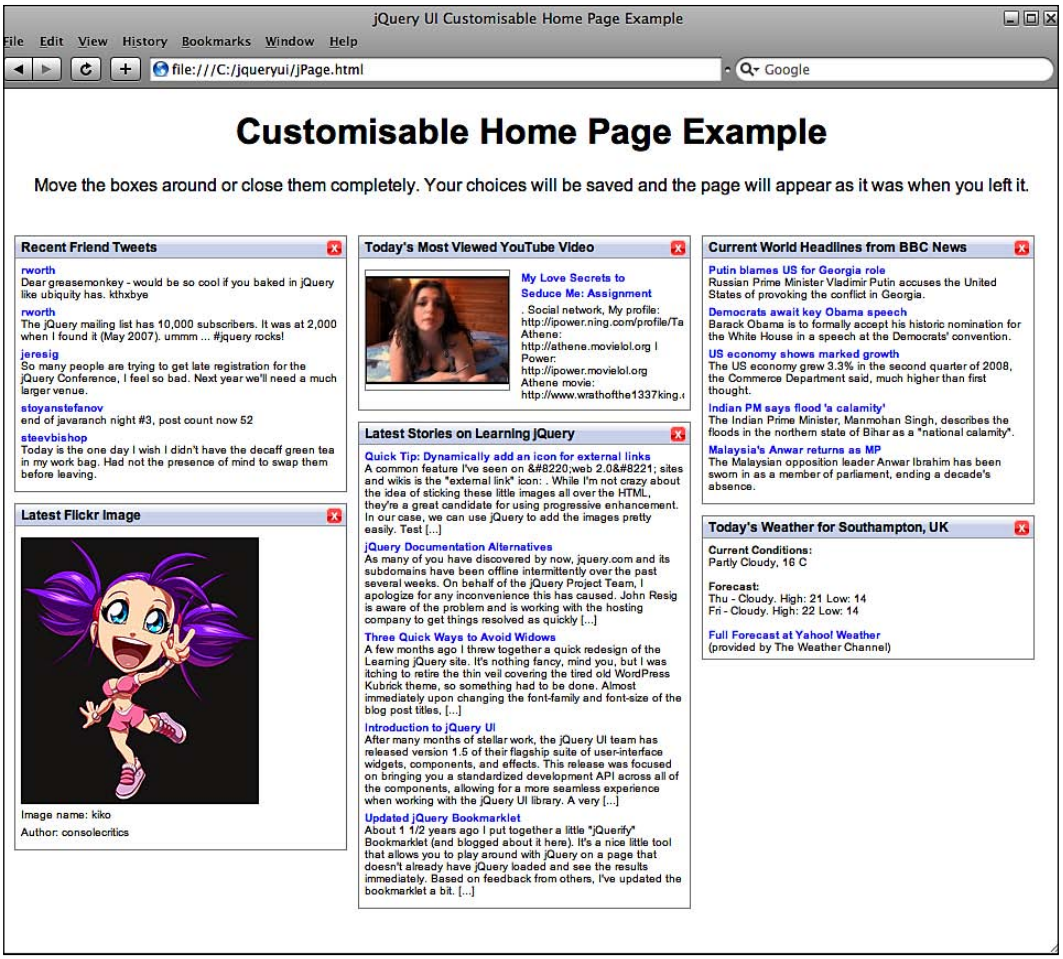
Sorting the tabs works on the `mousedown` event and selecting the tabs works on the `mouseup` event, so there are no event collisions and no situations arising where you want to select a tab but end up sorting it. The next screenshot shows how the tabs may appear after a sorting:




Unfortunately, IE has difficulties with this example. Although the tabs are sortable, any tab that is moved becomes unselectable, although by making use of tab callback functions, we could probably code around this.

# Fun with sortable

It's time for our final sortable example. We're going to put the component to good use by creating a page with content boxes on it that can be sorted into various positions to suit the visitors personal preference, a little like iGoogle. The following screenshot shows what we're aiming for:




The mark-up for the page is minimal as most of the content will be added dynamically from various remote sources. You don't need to worry about having a full web-server setup to complete this example. Most of the code uses JSON, which as you know can be interpreted directly in the browser. We'll also be making use of cookies, which again can be used purely with JavaScript.

To begin, create the following basic HTML page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/jPage.css">
```

```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Customizable Home Page Example</title>
  </head>
  <body>
    <div class="page">
      <h1>Customizable Home Page Example</h1>
      <p>Move the boxes around or close them completely. Your choices
will be saved and the page will appear as it was when you left it.<p>
        <a id="restore" href="#" title="Restore Deleted
Boxes">Restore Deleted Boxes</a>
      <div id="sortGrid">
        <div id="col1" class="col"></div>
        <div id="col2" class="col"></div>
        <div id="col3" class="col"></div>
        <div id="hidden"></div>
        <div class="clear"></div>
      </div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
ui.sortable.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery.cookie.js"></script>
  </body>
</html>
```

Save this as `jPage.html`. I said it would be simple, but let's just look at what the page contains. At the top, we've got a header, some explanatory text, and a link which will be used to reopen boxes that have been closed.

The main part of the page contains three `<div>` elements that will be styled to float next to each other to represent columns, plus a hidden column that will be used to store closed boxes. That's it, the rest of the elements are the `<script>` resources that we'll be using for this example.



The jQuery cookie plug-in by Klaus Hartl really helps us to avoid relying on back-end PHP (or other generic server-side environment) to process the desired state of the boxes. It also makes working with cookies much less cumbersome and saves us a good deal of code.

Next, we can add the CSS that is needed to make the page work. Some of the selectors in our stylesheet will be matching elements that don't yet exist, but we'll add the styling for them now anyway to save ourselves time later on. In a new page in your text editor, add the following code:

```
body { font-family:Arial, Helvetica, sans-serif; }
.page {
  width:960px; margin:auto; text-align:center;
  position:relative;
}
#sortGrid { width:960px; padding:20px 0; }
a#restore {
  font-family:Arial, Helvetica, sans-serif;
  position:absolute; right:44px; font-size:10px;
  color:#000000;
}
.col {
  float:left; width:312px; min-height:700px;
  height:auto !important; height:700px;
}
.box {
  width:290px; margin:0 0 10px 10px; position:relative;
  border:1px solid #999999; text-align:left;
  padding:25px 5px 5px 5px; font-size:10px;
  background-color:#ffffff;
}
.title {
  width:295px; height:20px; position:absolute; top:0; left:0;
  padding:3px 0 0 5px; font-size:12px; font-weight:bold;
  cursor:move;
  background:url(../img/jPage/titleBG.gif) repeat-x;
}
.close {
  width:15px; height:15px; position:absolute; right:3px;
  top:3px; background:url(../img/jPage/close.gif) no-repeat;
  cursor:pointer;
}
#hidden { display:none; }
.clear { clear:both; }
.box a {
  text-decoration:none; font-weight:bold; color:#3300ff;
}
#col2_youtube a { padding-left:0px; }
#video { overflow:hidden; }
.box p { margin:0 0 5px 0; padding:0; }
.box img { border:1px solid #000; margin:5px auto 2px auto; }
```

Save this as `jPage.css` in the `styles` folder. We'll just skim over the CSS as there are only a couple of points worth raising here.

One of the most salient points is the fact that we're using `min-height` on our columns. The reason for this is that if we don't set some kind of height on our columns they will collapse to nothing if all of the content boxes are moved out of them. Using `min-height` prevents this from happening and allows the columns to grow if a large box is moved into them. IE6 of course doesn't support `min-height`, hence the crafty hack.



We're using Dustin Diaz's celebrated Min-Height Fast Hack in our CSS for this example to improve the quality of the resulting page. For more information, see Dustin's blog at `http://www.dustindiaz.com/min-height-fast-hack/`

Other than this, the CSS merely lays out the page in the way we want. Near the end of the file, there are some rules that are used to specifically style the content that will be added to the individual boxes. The format of this content varies considerably depending on its source, so we have to use a couple of very specific rules to create the desired effect.

# The main script

To bring the page to life, we now need to focus on the JavaScript required to turn this collection of elements into a usable interface. Here's the code required in its entirety. Take a moment to look through it, or have the file from the code download at hand. We'll break it down into bite-sized pieces directly:

```
<script type="text/javascript">
  //function to execute when doc ready
  $(function() {
    //object of title names
    var titles = {};
    titles.twitter = "Recent Friend Tweets";
    titles.flickr = "Latest Flickr Image";
    titles.youtube = "Today's Most Viewed YouTube Video";
    titles.jquery = "Latest Stories on Learning jQuery";
    titles.worldNews = "Current World Headlines from BBC News";
    titles.weather = "Today's Weather for Southampton, UK";
    //check for cookie
    if (!$.cookie("columnOrder")) {
      //arrange default box layout
```

```
      $("<div>").addClass("box").attr("id", "col1_twitter")
.appendTo("#col1");
      $("<div>").addClass("title").attr("id", "twitterTitle")
.text(titles["twitter"]).appendTo("#col1_twitter");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#twitterTitle");
      $("<div>").addClass("box").attr("id", "col1_flickr")
.appendTo("#col1");
      $("<div>").addClass("title").attr("id", "flickrTitle")
.text(titles["flickr"]).appendTo("#col1_flickr");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#flickrTitle");

      $("<div>").addClass("box").attr("id", "col2_youtube")
.appendTo("#col2");
      $("<div>").addClass("title").attr("id", "youtubeTitle")
.text(titles["youtube"]).appendTo("#col2_youtube");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#youtubeTitle");
      $("<div>").addClass("box").attr("id", "col2_jquery")
.appendTo("#col2");
      $("<div>").addClass("title").attr("id", "jqueryTitle")
.text(titles["jquery"]).appendTo("#col2_jquery");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#jqueryTitle");

      $("<div>").addClass("box").attr("id", "col3_worldNews")
.appendTo("#col3");
      $("<div>").addClass("title").attr("id", "worldNewsTitle")
.text(titles["worldNews"]).appendTo("#col3_worldNews");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#worldNewsTitle");
      $("<div>").addClass("box").attr("id", "col3_weather")
.appendTo("#col3");
      $("<div>").addClass("title").attr("id", "weatherTitle")
.text(titles["weather"]).appendTo("#col3_weather");
      $("<div>").attr("title", "Close").addClass("close")
.appendTo("#weatherTitle");

      $("#hidden").empty();
    } else {

      //split serialized string
      var cols = $.cookie("columnOrder").split("&");

      for (var x = 0; x < cols.length; x++) {

        if(cols[x] != "") {

          //split the data string
          var col = cols[x].split("=")[0];
```

```
            var box = cols[x].split("=")[1];
             //build current box
            $("<div>").addClass("box").attr("id", col + "_" + box).
appendTo("#" + col);
            $("<div>").addClass("title").attr("id", box + "Title").
text(titles[box]).appendTo("#" + col + "_" + box);

$("<div>").attr("title","Close").addClass("close").appendTo("#" + box
+"Title");
            }
          }
        }
    //get twitter feed
    $.getJSON("http://pipes.yahoo.com/pipes/Sj5Zqa1q3RGsKtdWQBJ3AQ/
run?&_render=JSON&_callback=?", function(data) {
    for (var x = 0; x < 5; x++) {
      $("<a />").attr("href", "http://twitter.com/" + data.value.
items[x].description.split(":")[0]).text(data.value.items[x].
description.split(":")[0]).appendTo("#col1_twitter");
$("<p>").text(data.value.items[x].description.split(":")[1]).appendTo(
$("#twitterTitle").parent());
      }
    });
    //get most recent flickr image
 $.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?for
mat=json&jsoncallback=?", function(data){
    $("<a />").attr({"href":data.items[0].link,"id":"imgLink"}).append
To($("#flickrTitle").parent());
    $("<img />").attr("src", data.items[0].media.
m).appendTo("#imgLink");
    $("<p>").text("Image name: " + data.items[0].title).appendTo($("#f
lickrTitle").parent());
    $("<p>").text("Author: " + data.items[0].author.split("(")[1].
replace(")", "")).appendTo($("#flickrTitle").parent());
    });
    //get youtube vid
            $.getJSON("http://pipes.yahoo.com/pipes/Lt4yB_
hq3RGqImvDrLQIDg/run?&_render=JSON&_callback=?", function(data) {
      $("<div>").attr("id", "video").html("<div " + data.value.
items[0].description).appendTo($("#youtubeTitle").parent());
      $("#video div:first").css({width:290,paddingTop:5});
      $("#video div div:first").css({width:290});
      $("#video div table tbody tr:last").remove().prev().
css({width:300});
      $("#video div table tbody tr td:last").remove();
```

```
      $("#video div table tbody tr td:last").css({width:153});
      $("#video div table tbody tr td:last div a").css({fontSize:10});
      $("#video div table tbody tr td:last div:last").
css({fontSize:10});
      $("#video div table tbody tr:first td div").css({width:130,padd
ingTop:0});
    });

    //get learning jquery feed
            $.getJSON("http://pipes.yahoo.com/pipes/NjM4mhpr3RGN1_
VPPxJ3AQ/run?&_render=JSON&_callback=?", function(data) {
      for (var y = 0; y < 5; y++) {
        $("<a />").attr({"id":"articleLink","href":data.value.
items[y].link}).text(data.value.items[y].title).appendTo($("#jqueryTit
le").parent());
           $("<p>").text(data.value.items[y].description.
split("<")[0]).appendTo($("#jqueryTitle").parent());
       }
     });

     //get bbc news headlines
            $.getJSON("http://pipes.yahoo.com/pipes/Cm_
wLtdq3RGI5r2iBR50VA/run?&_render=JSON&_callback=?", function(data) {
      for (var x = 0; x < 5; x++) {
        $("<div>").addClass("headline").attr("id", "headline" + x).app
endTo($("#worldNewsTitle").parent());
        $("<a />").attr("href", data.value.items[x].link).text(data.
value.items[x].title).appendTo("#headline" + x);
                        $("<p>").text(data.value.items[x].
description).appendTo("#headline" + x);
       }
     });
      //get weather feed
            $.getJSON("http://pipes.yahoo.com/pipes/ZlCztwxr3RGvr976_
g6H4A/run?&_render=JSON&_callback=?", function(data) {
        $("<div>").attr("id", "weatherData").html(data.value.items[0].
description).appendTo($("#weatherTitle").parent());
        $("#weatherData img").remove();
        $("#weatherData br:first").remove();
      });

      //define config object
      var sortOpts = {
        handle: ".title",
        containment: "#sortGrid",
        dropOnEmpty: true,
        connectWith: ["#col1", "#col2", "#col3"],
```

```
        stop: function() {
          //serialze columns to get latest order
             var colOrders = $("#col1").sortable("serialize",
{key:"col1"}) + "&" + $("#col2").sortable("serialize", {key:"col2"})
+ "&" + $("#col3").sortable("serialize", {key:"col3"}) + "&" +
$("#hidden").sortable("serialize", {key:"hidden"});
          //write column order to cookie
          $.cookie("columnOrder", colOrders, { path:"/", expires:365
});
        }
      };
      //make columns sortable
      $("#col1").sortable(sortOpts);
      $("#col2").sortable(sortOpts);
      $("#col3").sortable(sortOpts);
      $("#hidden").sortable();

      //add closed item to hidden col and write new cookie
      $(".close").click(function() {
      $(this).parent().parent().appendTo("#hidden");
      //serialize columns to get latest order
      var colOrders = $("#col1").sortable("serialize", {key:"col1"})
+ "&" + $("#col2").sortable("serialize", {key:"col2"}) + "&" +
$("#col3").sortable("serialize", {key:"col3"}) + "&" + $("#hidden").
sortable("serialize", {key:"hidden"});
      //write column order to cookie
      $.cookie("columnOrder", colOrders, { path:"/", expires:365 });
    });
    //restore closed boxes
    $("#restore").click(function() {

      $("#hidden").children().each(function() {
        var col = "";
        //look for col with space
        $(".col").each(function() {
        ($(this).children().length < 2) ? col = $(this).attr("id") :
null ;
      });
        //get id and split to get box name
        var boxId = $(this).attr("id").split("_")[1];
        //add box to col
```

```
        $(this).appendTo("#" + col).attr("id", col + "_" + boxId);

        //serialze columns to get latest order
        var colOrders = $("#col1").sortable("serialize", {key:"col1"})
+ "&" + $("#col2").sortable("serialize", {key:"col2"}) + "&" +
$("#col3").sortable("serialize", {key:"col3"}) + "&" + $("#hidden").
sortable("serialize", {key:"hidden"});

        //write column order to cookie
        $.cookie("columnOrder", colOrders, { path:"/", expires:365 });
      });
    });
  });
</script>
```

I appreciate that sections of the above may have been a little unreadable. You may find it easier to look at the complete file in your text editor while reading this section. As I said, we'll break it down into its component parts now to see what each bit does.

One of the first things we do is create an object called `titles` which contains a series of strings that will form the title text of each box. This object is essential when we create the boxes according to the visitor's cookie instead of using the default layout, but we can also use it when creating the default layout too:

```
//object of title names
var titles = {};
titles.twitter = "Recent Friend Tweets";
titles.flickr = "Latest Flickr Image";
titles.youtube = "Today's Most Viewed YouTube Video";
titles.jquery = "Latest Stories on Learning jQuery";
titles.worldNews = "Current World Headlines from BBC News";
titles.weather = "Today's Weather for Southampton, UK";
```

Next, we have to see if the cookie indicating that the visitor has been to the page before and changed the box layout exists. If it doesn't exist, we go ahead and create the default boxes. There is a good deal of repetition in this section, but it would be more complicated if we tried to make some kind of factory function to produce the boxes because each box has a unique `id` and `title`:

```
//check for cookie
if (!$.cookie("columnOrder")) {

  //arrange default box layout
  $("<div>").addClass("box").attr("id", "col1_twitter").
appendTo("#col1");
  $("<div>").addClass("title").attr("id", "twitterTitle").
text(titles["twitter"]).appendTo("#col1_twitter");
```

```
  $("<div>").attr("title", "Close").addClass("close").appendTo("#twit
terTitle");
  $("<div>").addClass("box").attr("id", "col1_flickr")
.appendTo("#col1");
  $("<div>").addClass("title").attr("id", "flickrTitle")
.text(titles["flickr"]).appendTo("#col1_flickr");
  $("<div>").attr("title", "Close").addClass("close").appendTo("#flic
krTitle");
  $("<div>").addClass("box").attr("id", "col2_youtube")
.appendTo("#col2");
  $("<div>").addClass("title").attr("id", "youtubeTitle")
.text(titles["youtube"]).appendTo("#col2_youtube");
  $("<div>").attr("title", "Close").addClass("close").appendTo("#yout
ubeTitle");
  $("<div>").addClass("box").attr("id", "col2_jquery")
.appendTo("#col2");
  $("<div>").addClass("title").attr("id", "jqueryTitle")
.text(titles["jquery"]).appendTo("#col2_jquery");
  $("<div>").attr("title", "Close").addClass("close").appendTo("#jque
ryTitle");
  $("<div>").addClass("box").attr("id", "col3_worldNews")
.appendTo("#col3");
  $("<div>").addClass("title").attr("id", "worldNewsTitle").text(title
s["worldNews"]).appendTo("#col3_worldNews");
  $("<div>").attr("title", "Close").addClass("close").appendTo("#worl
dNewsTitle");
  $("<div>").addClass("box").attr("id", "col3_weather")
.appendTo("#col3");
  $("<div>").addClass("title").attr("id", "weatherTitle")
.text(titles["weather"]).appendTo("#col3_weather");
  $("<div>").attr("title", "Close").addClass("close").appendTo("#weath
erTitle");
  $("#hidden").empty();
} else {
```

The next part of the `if` statement is executed if the visitor has used the page before and this time we do have a function that churns out the boxes. We can do this because all of the dynamic information that we need can be extracted from the cookie:

```
//split serialized string
var cols = $.cookie("columnOrder").split("&");
  for (var x = 0; x < cols.length; x++) {
    if(cols[x] != "") {
```

```
        //split the data string
        var col = cols[x].split("=")[0];
        var box = cols[x].split("=")[1];

        //build current box
        $("<div>").addClass("box").attr("id", col + "_" + box).
appendTo("#" + col);
        $("<div>").addClass("title").attr("id", box + "Title").
text(titles[box]).appendTo("#" + col + "_" + box);
      $("<div>").attr("title","Close").addClass("close").appendTo("#" +
box +"Title");
      }
    }
}
```

Thanks to the `serialize` method which we use later in the script, the data that the cookie contains will be delimited, or separated, with ampersands so the above bit of code first creates a collection by splitting the string on `&` characters.

We then use a `for` loop to iterate through this collection and for each item we perform another split. The format of each item will be *columnName*=*sortableName*, so this time the split is on the `=` character to separate the column and box `id` for each box. We then build the box, in a very similar way as the default boxes, but this time using either the column or box name to generate the required `ids`.

Our `titles` object comes in handy here, thanks to the fact that objects in JavaScript can use dot notation and bracket notation, allowing us to use a variable to extract the correct `title` text. Once constructed, each box is appended to the column it was placed in by the visitor.

In the next part of the script, we obtain all of our JSON data from each of the different sources. We do this using jQuery's `getJSON` method, which I'm hoping you'll already be familiar with:

```
//get twitter feed
$.getJSON("http://pipes.yahoo.com/pipes/Sj5Zqa1q3RGsKtdWQBJ3AQ/
run?&_render=JSON&_callback=?", function(data) {
  for (var x = 0; x < 5; x++) {
    $("<a />").attr("href", "http://twitter.com/" + data.value.
items[x].description.split(":")[0]).text(data.value.items[x].
description.split(":")[0]).appendTo("#col1_twitter");$("<p>").
text(data.value.items[x].description.split(":")[1]).appendTo($("#twitt
erTitle").parent());
  }
});

//get most recent flickr image
```

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?form
at=json&jsoncallback=?", function(data){
  $("<a />").attr({"href":data.items[0].link,"id":"imgLink"}).appendTo
($("#flickrTitle").parent());
  $("<img />").attr("src", data.items[0].media.
m).appendTo("#imgLink");
  $("<p>").text("Image name: " + data.items[0].title).appendTo($("#fli
ckrTitle").parent());
  $("<p>").text("Author: " + data.items[0].author.split("(")[1]
.replace(")", "")).appendTo($("flickrTitle").parent());
});

//get youtube vid
$.getJSON("http://pipes.yahoo.com/pipes/Lt4yB_hq3RGqImvDrLQIDg/run?&_
render=JSON&_callback=?", function(data) {
  $("<div>").attr("id", "video").html("<div " + data.value.items[0]
.description).appendTo($("#youtubeTitle").parent());
  $("#video div:first").css({width:290,paddingTop:5});
  $("#video div div:first").css({width:290});
  $("#video div table tbody tr:last").remove().prev()
.css({width:300});
  $("#video div table tbody tr td:last").remove();
  $("#video div table tbody tr td:last").css({width:153});
  $("#video div table tbody tr td:last div a").css({fontSize:10});
  $("#video div table tbody tr td:last div:last").css({fontSize:10});
  $("#video div table tbody tr:first td div").css({width:130,
paddingTop:0});
});

//get learning jquery feed
$.getJSON("http://pipes.yahoo.com/pipes/NjM4mhpr3RGN1_VPPxJ3AQ/run?&_
render=JSON&_callback=?", function(data) {
  for (var y = 0; y < 5; y++) {
    $("<a />").attr({"id":"articleLink","href":data.value.items[y]
.link}).text(data.value.items[y].title).appendTo($("#jqueryTitle")
.parent());
      $("<p>").text(data.value.items[y].description.split("<")[0])
.appendTo($("#jqueryTitle").parent());
  }
});

//get bbc news headlines
$.getJSON("http://pipes.yahoo.com/pipes/Cm_wLtdq3RGI5r2iBR50VA/
run?&_render=JSON&_callback=?", function(data) {
  for (var x = 0; x < 5; x++) {
    $("<div>").addClass("headline").attr("id", "headline" + x)
.appendTo($("#worldNewsTitle").parent());
```

```
    $("<a />").attr("href", data.value.items[x].link).text(data.value.
items[x].title).appendTo("#headline" + x);
      $("<p>").text(data.value.items[x].description)
.appendTo("#headline" + x);
  }
});

//get weather feed
$.getJSON("http://pipes.yahoo.com/pipes/ZlCztwxr3RGvr976_g6H4A/
run?&_render=JSON&_callback=?", function(data) {
  $("<div>").attr("id", "weatherData").html(data.value.items[0]
.description).appendTo($("#weatherTitle").parent());
  $("#weatherData img").remove();
  $("#weatherData br:first").remove();
});
```

Most of the code in this section deals with getting the data we want from the JSON objects and constructing the necessary elements used to display the data. These elements are usually `<div>`, `<p>`, `<img>`, or `<a>` elements.

Again, because the format of each data object is different, there's no simple way of reducing this large block of code. Not all of the remote data sources we're working with return data in JSON format so we pipe that data in via Yahoo! Pipes instead.

Sometimes we strip out the data we want and create our own elements to hold the data. At other times, we use existing mark-up within the returned data structure. Some of the mark-up returned in each JSON object is, frankly, shocking, hence why we sometimes have to strip out images, breaks, or table cells.



We've used Yahoo Pipes in this example to convert standard RSS feeds into JSON. This is because although JSON is an efficient and lightweight method of data interchange, which when combined with AJAX negates the need for server-side processing, it has yet to proliferate in the way that RSS has.

Next, we define the configuration object used to make the sortables interaction helper work in the way we want it to:

```
//define config object
var sortOpts = {
  handle: ".title",
  containment: "#sortGrid",
  dropOnEmpty: true,
  connectWith: ["#col1", "#col2", "#col3"],
  stop: function() {
    //serialze columns to get latest order
```

```
    var colOrders = $("#col1").sortable("serialize", {key:"col1"})
+ "&" + $("#col2").sortable("serialize", {key:"col2"}) + "&" +
$("#col3").sortable("serialize", {key:"col3"}) + "&" + $("#hidden")
.sortable("serialize", {key:"hidden"});

    //write column order to cookie
    $.cookie("columnOrder", colOrders, { path:"/", expires:365 });
  }
};
```

Most of these properties we've looked at before. The most important one is the stop custom event handler which serializes the column order following a sort interaction and writes a new cookie containing the layout of the boxes. We also make sure that each of the columns (except the `hidden` column) is connected so that the boxes can move freely between them.

To make the boxes sortable, we simply call the `sortable` method passing in the configuration object we just created:

```
//make columns sortable
$("#col1").sortable(sortOpts);
$("#col2").sortable(sortOpts);
$("#col3").sortable(sortOpts);
$("#hidden").sortable();
```

Next we add the function that closes boxes and moves them to the hidden column:

```
//add closed item to hidden col and write new cookie
$(".close").click(function() {
  $(this).parent().parent().appendTo("#hidden");

  //serialze columns to get latest order
  var colOrders = $("#col1").sortable("serialize", {key:"col1"})
+ "&" + $("#col2").sortable("serialize", {key:"col2"}) + "&" +
$("#col3").sortable("serialize", {key:"col3"}) + "&" + $("#hidden")
.sortable("serialize", {key:"hidden"});

  //write column order to cookie
  $.cookie("columnOrder", colOrders, { path:"/", expires:365     });
});
```

First, the box that has been closed by clicking the close icon is appended to the hidden column and then the new sortable order is serialized and written to the cookie again. We have to do this because the `stop` event handler is not triggered as nothing is actually sorted.

Our final function deals with restoring the closed boxes when the **restore** link at the top of the page is clicked:

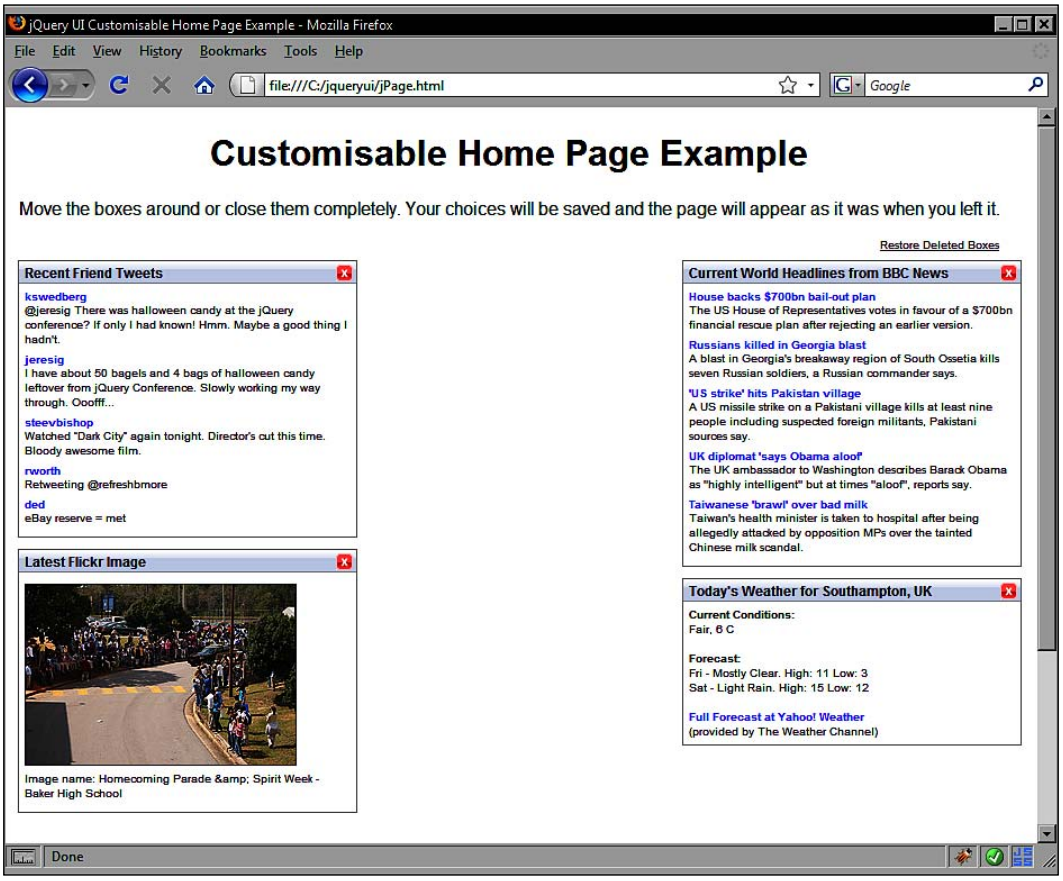
```
//restore closed boxes
$("#restore").click(function() {

  $("#hidden").children().each(function() {

    var col = "";

    //look for col with space
    $(".col").each(function() {
    ($(this).children().length < 2) ? col = $(this).attr("id") : null
;
  });

  //get id and split to get box name
  var boxId = $(this).attr("id").split("_")[1];

  //add box to col
  $(this).appendTo("#" + col).attr("id", col + "_" + boxId);

  //serialze columns to get latest order
   var colOrders = $("#col1").sortable("serialize", {key:"col1"})
+ "&" + $("#col2").sortable("serialize", {key:"col2"}) + "&" +
$("#col3").sortable("serialize", {key:"col3"}) + "&" + $("#hidden")
.sortable("serialize", {key:"hidden"});

   //write column order to cookie
   $.cookie("columnOrder", colOrders, { path:"/", expires:365 });

});
```

The first part of the function cycles through each column to see which ones have space for another box. The number of boxes in each column will depend on what the visitor has moved or closed previously so we can't make any assumptions about where to put the box.

Boxes moved to the hidden column will automatically be renamed when the page loads so that they have the `id` `hidden_`*sortableName*. We need to change this so that the boxes `id` is *columnName_sortableName* instead.

We then add the box to a column that has space, setting its `id` in the process before finally serializing the new column order and writing a new cookie once more, which brings us to the end of the example.

You should find when you run the page that you can move the boxes around, close your browser, run the page again, and see the boxes retain the order that you gave them:




Please note that that example does not run correctly in Opera and exhibits the same unusual placement of sorted items that occurred in some of the earlier connected-list examples in the chapter.

# Summary

We've finished off our tour of the interaction components of the library by looking at the sortables component. Like the other modules that we looked at before, it has a wide range of properties and methods that allow us to configure and control its behavior and appearance in both simple and more complex implementations.

We started the chapter off with a look at a simple, default implementation with no configuration to see the most basic level of functionality added by the component. We looked at some of the different elements that can be made sortable and added some basic styling to the page.

Following this, we looked at the range of configurable properties that are exposed by the sortable API. The list is extensive and provides a wide range of functionality that can be enabled or disabled with ease.

We moved on to look at the extensive event model used by this component which gives us the ability to react to different events as they occur in any sort operation initiated by the visitor.

Connected lists offer the ability to be able to exchange sortable items, giving our visitors the ability to move items between separate lists. We saw the additional properties and events that are used specifically with connected sortable lists.

In the last part of the chapter, we looked at the methods available for use with the sortables component and focused on the highly useful `serialize` method, and also had a quick look at its compatibility with other members of the jQuery UI library in the form of the sortable tabs example.

# 12
# UI Effects

So far, we've looked at a range of incredibly useful widgets and extension helpers. All are easy to use, but some have had their subtle nuances which have required consideration and thought during their use. Some of the components' APIs that we have looked at have been huge. The effects of the library on the other hand are for the most part, extremely compact, with very few properties to learn and no methods at all. Using the effects is as simple as calling the effect's constructor and including maybe one or two properties if required.

Each of the previous chapters has finished on a *fun with* section in which the API that the chapter focuses on has been put to use in a functional, potentially useful, and above all fun scenario. This chapter isn't going to end with a *fun with* example – the whole chapter is instead going to be written from a 'fun' perspective, so we can sit back, relax a little, and enjoy the show that the UI effect components can put on for us.

The effects that we'll be looking at in this chapter are as follows:

- blind
- bounce
- clip
- drop
- explode
- fold
- highlight
- pulsate
- puff
- scale
- shake
- slide
- transfer

# The core effects file

Like the individual components themselves, the effects require the services of a separate core file which provides essential services to the effects, such as creating wrapper elements, and controlling the animations. Most, but not all, of the effects have their own source file.

All we need to do to use an effect is include the core file (`effects.core.js`) in the page before the effect's source file, and then forget about it. Unlike the `ui.core.js` file however, the `effects.core.js` file has been designed to be used, in part, completely standalone.

When using the core effect file on its own we can take advantage of color animations, such as smoothly changing the background color of an element into another color (and not just a snap change but a smooth morphing of one color into another), class transitions, and advanced easing animations.

# Color animations

Let's look at color animations first. These are very easy to implement and give an attractive result quickly. Create the following new page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
colorAnim.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Color Animation Example</title>
  </head>
  <body>
    <div><label>Name: </label><input type="text"></div>
    <div><label>Age: </label><input type="text"></div>
    <div><label>Email: </label><input type="text"></div>
    <button id="submit">Submit</button>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
```

```
        $("#submit").click(function() {
          //check fields not empty
          $("input").each(function() {
            //color red if they are
            ($(this).val().length == 0) ? $(this).animate({
              backgroundColor:"#ff9999",
              borderTopColor:"#ff0000",
              borderRightColor:"#ff0000",
              borderBottomColor:"#ff0000",
              borderLeftColor:"#ff0000"
            }) : $(this).animate({ //color green if not
              backgroundColor:"#ccffcc",
              borderTopColor:"#00ff00",
              borderRightColor:"#00ff00",
              borderBottomColor:"#00ff00",
              borderLeftColor:"#00ff00"
            });
          });
        });
      </script>
    </body>
  </html>
```

Save the page as `colorAnim.html`. As you can see, all we need are jQuery and the `effects.core.js` file. When I said the `effects.core.js` file could be used standalone, I actually meant on top of the normal jQuery library, although it is still standalone as far as the UI effects are concerned.

The `animate` method, as I'm sure you're aware, is part of jQuery rather than jQuery UI, but the `effects.core.js` file extends the `animate` method by allowing it to specifically work with colors and classes.

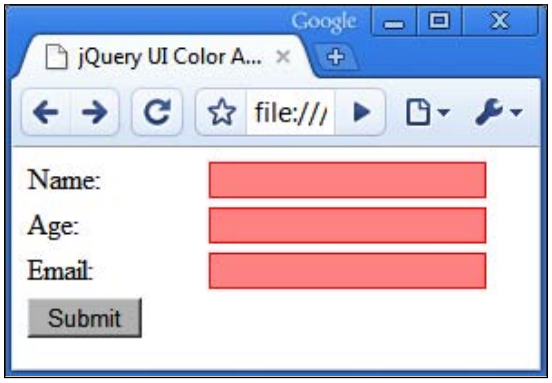
When the **Submit** `<button>` is clicked in this example, we simply use the `animate` method to apply a series of new CSS properties to the target elements. These style properties are supplied to the method as the properties and values of a literal object.

We also use a basic stylesheet in this example. In another new page, add the following;

```
div { margin-bottom:5px; }
label { display:block; width:100px; float:left; }
input { border:1px solid #000000; }
```

Save this as `colorAnim.css` in the `styles` folder. When we view this page in our browser, we should see that any fields left blank smoothly turn red when the **Submit** `<button>` is clicked, while fields that are not empty smoothly turn green. The most attractive however are when a field changes from red to green.

The following screenshot shows the page once the **Submit** `<button>` has been clicked:




The style attributes that color animations can be used on are:

- `backgroundColor`
- any `borderColor`
- `color`
- `outlineColor`

Colors may be specified using either RGB or HEX, or even standard color names.

# Class transitions

In addition to animating individual color attributes, `effects.core.js` also gives us the powerful ability to animate between entire classes, allowing us to switch styles smoothly and seamlessly without sudden, jarring changes. Let's look at this aspect of the file's use in the following example. Create the following new file in your text editor:

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/
classAnim.css">
```

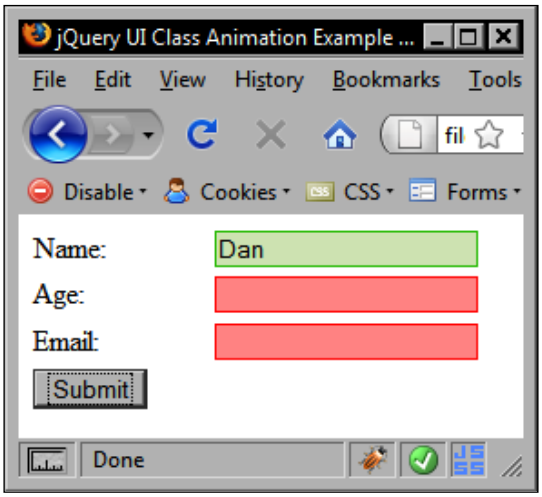
```
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Class Animation Example</title>
  </head>
  <body>
    <div><label>Name: </label><input type="text"></div>
    <div><label>Age: </label><input type="text"></div>
    <div><label>Email: </label><input type="text"></div>
    <button id="submit">Submit</button>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

      //add click handler for button
      $("#submit").click(function() {

        //check fields
        $("input").each(function() {

        //if input already has error class
        if ($(this).hasClass("error")) {

          //do nothing if empty or add pass class
          ($(this).val().length == 0) ? null : $(this).
switchClass("error", "pass", 2000);

          //if input already has pass class
          } else if ($(this).hasClass("pass")) {

            //do nothing if not empty or add error class
            ($(this).val().length != 0) ? null : $(this).
switchClass("pass", "error", 2000);

          //if has neither class
          } else {

            //add error class if empty, pass if not
            ($(this).val().length == 0) ? $(this).addClass("error",
2000) : $(this).addClass("pass", 2000);
          }
          });
        });
      });
    </script>
  </body>
</html>
```

Save this as `classAnim.html`. The `effects.core.js` extends the jQuery `addClass` method by allowing us to specify a duration over which the new class name should be applied instead of just switching it instantly. We also use the `switchClass` method of the `effects.core.js` file when the fields already have one of the class names.

Essentially, the page functions as it did before, although using this type of class transition allows us to use non-color-based style rules as well, so we could adjust widths, heights, or anything else controlled by CSS. As in the previous example, we have a stylesheet attached. This is essentially the same as in the previous example except with some styles for our two new classes. Add the following selectors and rules to `colorAnim.css`:

```
.error { border:1px solid #ff0000; background-color:#ff9999; }
.pass { border:1px solid #00ff00; background-color:#ccffcc; }
```

Save the updated file as `classAnim.css` in the `styles` folder. In the next screenshot, we see the page after it has been interacted with:




Please note that at the time of writing, this example only works in Firefox.

# Advanced easing

The `animate` method found in standard jQuery has some basic easing capabilities built in, but for more advanced easing, you had to include an additional easing plug-in (ported to jQuery by GSGD).

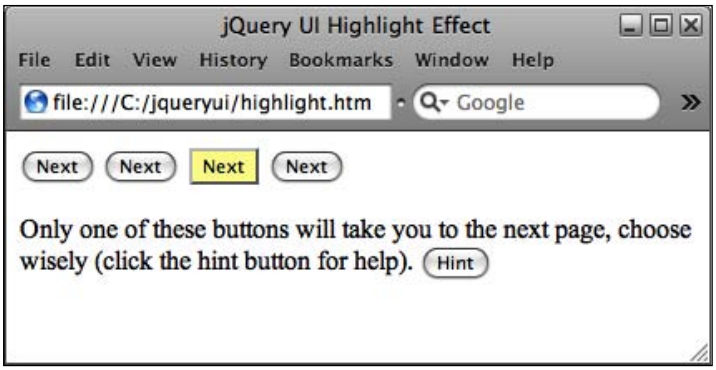
The `effect.core.js` file however has all of these advanced easing options built right in so there is no need to include additional plugins. We won't be looking at them in any real detail in this section, however, as we'll be using them in some of the examples later on in the chapter.

# Highlighting

The highlight effect temporarily applies a light yellow coloring to any element that it's called on. Let's put a simple example together so we can see the effect in action. In a new page in your text editor, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Highlight Effect</title>
  </head>
  <body>
    <button id="a">Next</button>
    <button id="b">Next</button>
    <button id="c">Next</button>
    <button id="d">Next</button>
    <p>Only one of these buttons will take you to the next page,
choose wisely (click the hint button for help).
    <button id="Hint">Hint</button>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.highlight.js"></script>
<script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        $("#hint").click(function() {
          //highlight specified element
          $("#c").effect("highlight");
        });
      });
    </script>
  </body>
</html>
```

Save this as `highlight.html`. The code which invokes the highlight effect takes the same familiar form as other library components. The `effect` constructor is called and the actual effect is specified as a string argument in the constructor. View the example and click the **Hint** button. The third button should briefly be highlighted:




The library files we needed for this example are listed below:

- `jquery-1.2.6.js`
- `effects.core.js`
- `effects.highlight.js`

While our example may seem a little contrived, it is easy to see the potential for this effect as an assistance tool on the front-end. Whenever there is a sequence of actions that needs to be completed in a specific order, the `highlight` effect can instantly give the visitor a visual cue as to the step that needs to be completed next. Similarly, it could be used in a tutorial or electronic manual to draw attention to a particular part of the screen.

# Additional effect parameters

Each of the `effect` constructors, as well as the parameter which dictates which effect is actually applied, can take up three additional parameters which control how the effect functions. All are optional, and consist of the following (in the listed order):

- An object containing additional configuration properties
- An integer representing in milliseconds the duration of the effect, or a string specifying one of `slow`, `normal`, or `fast`.
- A callback function that is executed when the effect ends

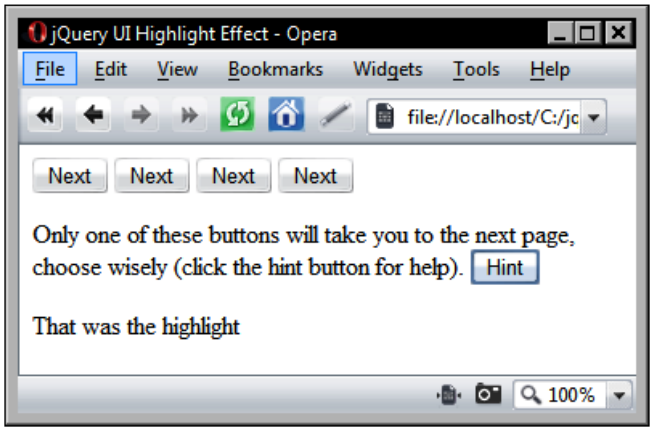
Let's add these additional parameters into our highlight example to clarify their usage. Change the final `<script>` element in `highlight.html` so that it appears as follows:

```
<script type="text/javascript">
  //function to execute when doc ready
  $(function() {
    $("#hint").click(function() {
      //highlight specified element
      $("#c").effect("highlight", {}, 2000, function() {
        $("<p>").text("That was the highlight").appendTo("body");
});
    });
  });
</script>
```

Save this as `highlightParameter.html`. Perhaps the most striking feature of our new code is the empty object passed as the second argument. In this example, we don't need any additional configurable properties, but we still need to pass in the empty object in order to access the third and fourth arguments.

The `highlight` effect has only one configurable property that can be used in the configuration object passed and that is the highlight color.

The animation should now proceed much slower as we have set the duration to `2000` milliseconds (2 seconds). Note that this third parameter may also take a string representing the speed of the animation. Our callback function, passed as the fourth and final argument, is perhaps the least useful callback in the history of JavaScript, but it does serve to illustrate how easy it is to arrange additional post-animation code execution. Here's how the page should look after the **Hint** button has been clicked:



# Bouncing

Another simple effect we can use with little configuration is the bounce effect. To see this effect in action create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/bounce.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Bounce Effect</title>
  </head>
  <body>
    <div id="menu">
      <a href="#">Home</a><a href="#">About</a><a href="#">Help</a>
<a href="#">Products</a><a href="#">Services</a>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.bounce.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //bounce the link on mouseover
        $("#menu a").mouseover(function() {
          $(this).effect("bounce");
        });
      });
    </script>
  </body>
</html>
```

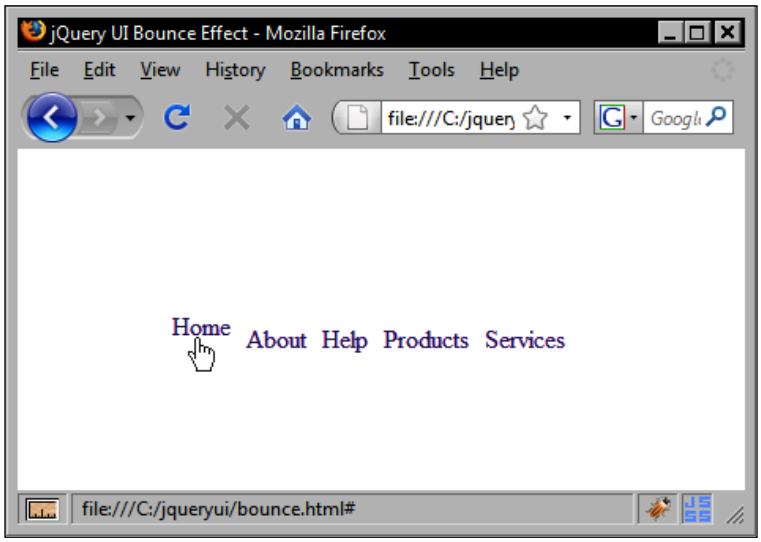
Save this as `bounce.html`. Using the bounce effect in this example shows how easy it is to add this simple but attractive effect. We didn't use any of the effect's configuration properties in this example, but the bounce effect does have three that may be useful in other situations:

| Property | Default | Usage |
|---|---|---|
| `direction` | `up` | Sets the direction of the bounce |
| `distance` | `20` (pixels) | Sets the distance of the first bounce |
| `times` | `5` | Sets the number of times the element should bounce |

You'll notice when you run the example that the bounce effect has an ease-out easing feature built into it, so the distance of the bounce will automatically decrease as the animation transpires. We also need a little CSS for this example. Add the following styles in a new page:

```
#menu {
  position:relative; top:100px; width:275px; margin:0 auto;
}
#menu a {
  float:left; height:20px; padding:2px 5px;
  text-decoration:none;
}
```

Save this as `bounce.css` in the `styles` folder. Here's how the page should look:




# Shaking

The `shake` effect is similar to the bounce effect but with the crucial difference of not having any built-in easing, so the targeted element will shake the same distance for the specified number of times instead of lessening each time (although it will come to a smooth stop at the end of the animation).

Let's change the previous example so that it uses the `shake` effect instead of the `bounce` effect. Change `bounce.html` so that it appears as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/bounce.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Shake Effect</title>
  </head>
  <body>
    <div id="menu">
      <a href="#">Home</a><a href="#">About</a><a href="#">Help</a>
<a href="#">Products</a><a href="#">Services</a>
    </div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.shake.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        //shake the link on mouseover
        $("#menu a").mouseover(function() {
          $(this).effect("shake", {direction:"up"}, "fast" );
        });
      });
    </script>
  </body>
</html>
```

Save this as `shake.html`. This time as well as changing the effect we've also made use of one of the configuration properties, the `direction` property, which controls the direction of the shake. This is to override the default setting for this property which is `left`.

This effect shares the same properties as the bounce effect, although the defaults are set slightly differently. The properties are listed in the following table:

| Property | Default | Usage |
|---|---|---|
| `direction` | `left` | Sets the direction of the shake |
| `distance` | `20` (pixels) | Sets the distance of the shake |
| `times` | `3` | Sets the number of times the element should shake |

# Transference

The transfer effect is different from the others in that it doesn't directly affect the targeted element. Instead, it transfers the outline of a specified element to another specified element. To see this effect in action, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/transfer.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Transfer Effect</title>
  </head>
  <body>
    <div id="container">
      <div id="productContainer">
        <img alt="GTX 280" src="img/gcard.jpg"></img><p>BFG GTX 280 OC
1GB GDDR3 Dual DVI HDTV Out PCI-E Graphics Card</p><p id="price">Cost:
$350</p><div id="purchase"><button id="buy">Buy</button></div>
      </div>
      <div id="basketContainer">
        <div id="basket"></div>
        <p>Basket total: <span id="total">0</span></p>
      </div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.transfer.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
```

```
        $(function() {
          $("#buy").click(function() {
            $("#productContainer img").effect("transfer", { to:"#basket"
}, 750, function() {
              var currentTotal = $("#total").text();
              numeric = parseInt(currentTotal);
              $("#total").text(numeric + 1);
            });
          });
        });
    </script>
  </body>
</html>
```

Save this as `transfer.html`. We've created a basic product listing for an imaginary hardware retailer. When the **Buy** `<button>` is clicked, the transfer effect will give the impression of the product being moved into the basket.

Of course, a proper shopping cart application would be exponentially more complex than this, but we do get to see the transfer effect and get to use the built-in callback function to do a little post-animation processing, so the exercise should still be beneficial.

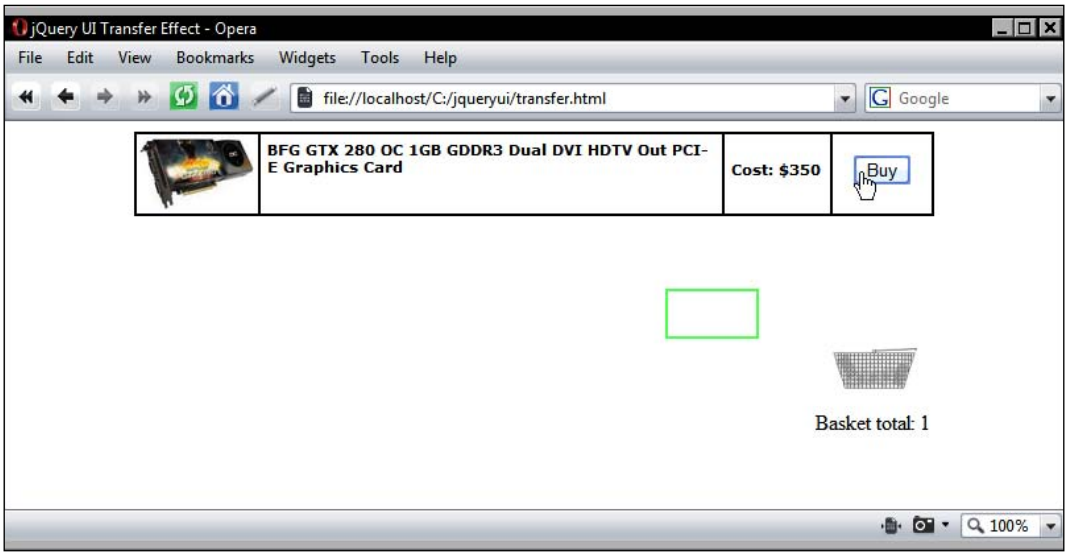
We also need some CSS for this example, so create the following new stylesheet:

```
#container { width:607px; margin:0 auto; }
#productContainer img {
  width:92px; height:60px;
  border:2px solid #000000;
  float:left; position:relative;
}
#productContainer p {
  width:340px; height:50px;
  font-family:Verdana; font-size:11px; font-weight:bold;
  float:left;
  margin:0; padding:5px;
  border-top:2px solid #000000;
  border-right:2px solid #000000;
  border-bottom:2px solid #000000;
}
p#price {
  height:35px; width:70px;
  padding-top:20px; float:left;
}
#purchase {
```

```
    height:44px; width:75px;
    border-top:2px solid #000000;
    border-right:2px solid #000000;
    border-bottom:2px solid #000000;
    padding-top:16px; float:left;
    text-align:center;
  }
  #basketContainer {
    float:right; width:90px; margin-top:100px;
  }
  #basket {
    width:65px; height:31px;
    position:relative; left:13px;
    background:url(../img/basket.gif) no-repeat;
  }
  .ui-effects-transfer { border:2px solid #66ff66; }
```

Save this as `transfer.css` in the `styles` folder. The key rule in our stylesheet is the one which targets the element that has a `class` of `ui-effects-transfer`. This element is created by the control and together with our styling produces the actual effect.

Run the file in your browser. I think you'll agree that it's a nice effect which would add value to any page that it was used on. Here's how it should look while the transfer is occurring:



The transfer effect has just two configurable properties, one of which is required and that we have already seen. For reference, both are listed in the following table:

| Property | Default | Usage |
|---|---|---|
| `className` | `ui-effects-transfer` | A new class to apply to the element the transfer originates from |
| `to` | *none* | Sets the element the effect will be transferred to. This property is mandatory |

The four effects that we've looked at so far all have one thing in common—they can only be used with the `effect` method. The remaining effects can be used not only with the `effect` method, but also with the `toggle`, and the `show`/`hide` methods. Let's take a look.

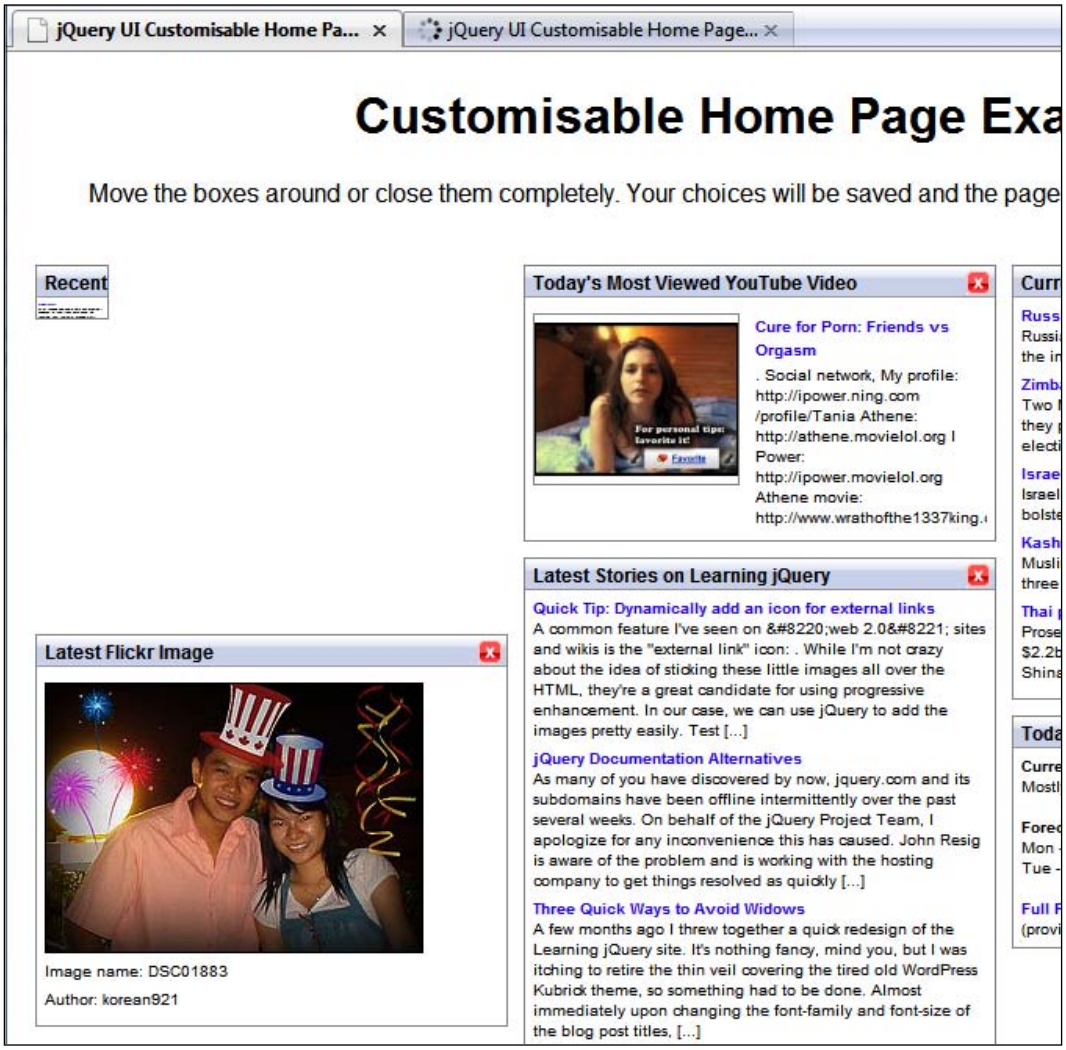
# Scaling

The next effect that we'll look at is scaling, which allows us to shrink or grow any specified element. At the end of the last chapter, we created a page that had a series of boxes on it which could be reordered or closed. When they were closed, they simply vanished instantly from the page.

Let's use the `scale` effect to make them gracefully shrink to nothing instead. Change the anonymous click function attached to the `.close` button in `jPage.html` so that it appears as follows:

```
//close the box on close icon click
$(".close").click(function() {

  //shrink the box to nothing then append to closed list
  $(this).parent().parent().effect("scale", { percent:0 }, "slow",
function() {
    $(this).appendTo("#hidden");
  });
});
```

Save this file as `scaling.html`. The `percent` property indicates the ending size of the element the effect is applied to. Here's how one of our boxes should look in mid-scale:



There are several more properties that can be used with `scale`, which are as follows:

| Property | Default | Usage |
|---|---|---|
| `direction` | `both` | Sets the direction to scale the element in. May be a string specifying either `both`, `vertical`, or `horizontal` |
| `from` | `{}` | Sets the starting `height` and `width` of the element to be scaled |
| `origin` | `["middle","center"]` | Sets the vanishing point, used with `show`/`hide` animations |
| `percent` | `0` | Sets the end size of the scaled element |

I mentioned a little while ago that the effects that we're looking at now can be used with other methods. The file in our previous example could be reconstructed to use the `hide` method instead:

```
//close the box on close icon click
$(".close").click(function() {

  //shrink the box to nothing then append to closed list
  $(this).parent().parent().hide("scale", { }, "slow", function() {
    $(this).appendTo("#hidden");
  });
});
```

Save this variation as `scalingHide.html`. We've gotten away with a slightly lighter method as we don't have to supply the `percent` property in our configuration object, but other than this, the effects are very similar code-wise. Visually, the only difference in the execution of this version of the file is that the boxes now vanish to the center instead of the top-left.

# Element explosion

The explosion effect is truly awesome. The targeted element is literally exploded into a specified number of pieces before disappearing completely. It's an easy effect to use and has few configuration properties but the visual impact of this effect is huge, giving you a lot of effect in return for very little code. Let's see a basic example. Create the following new page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
```

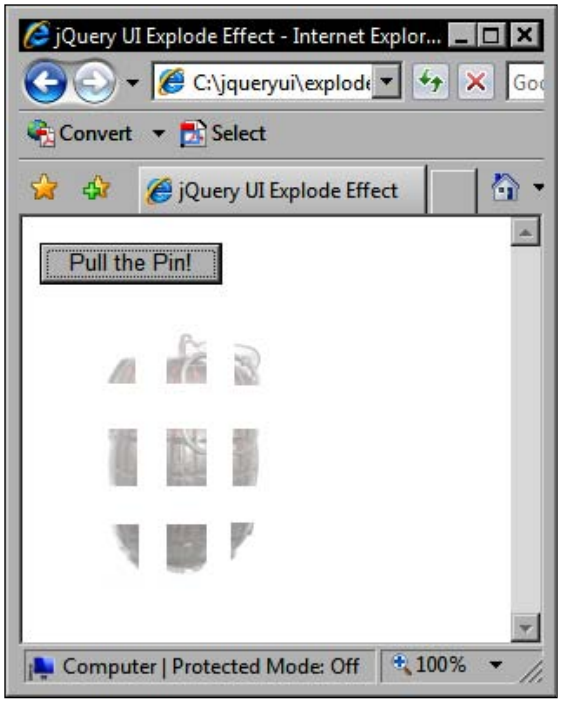
```
    <link rel="stylesheet" type="text/css" href="styles/explode.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Explode Effect</title>
  </head>
  <body>
    <button id="detonate">Pull the Pin!</button>
    <div id="theBomb"></div>

    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.explode.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {

        $("#detonate").click(function() {
          $("#theBomb").effect("explode");
        });
      });
    </script>
  </body>
</html>
```

Save this as `explode.html`. We also need a little CSS. Create the following stylesheet:

```
#theBomb {
  width:69px; height:100px;
  background:url(../img/nade.jpg) no-repeat;
  position:absolute;
  top:50px; left:50px;
}
```

Save this as `explode.css` in the `styles` folder. As you can see, the code is extremely simple and can be used completely out of the box with no additional configuration. This effect has only one configurable property, which is the `pieces` property and determines how many pieces the element is exploded into. The default is `9`.

As our example shows, the effect can be used with either simple CSS properties like coloured backgrounds and borders, or more complex implementations involving proper images:




Physicists sometimes speculate as to why the arrow of time seems to only point forwards. They invariably ask themselves philosophical questions like 'why do we not see grenades spontaneously forming from a large cloud of debris?' (actually, the object is usually an egg but I don't think an egg-based example would have had the same impact!)

jQuery UI cannot help our understanding of entropy, but it can show us what a grenade spontaneously reassembling might look like. Change the click handler in the previous function so that it appears as follows:

```
//show the image
$("#detonate").click(function() {
  $("#theBomb").show("explode");
});
```

Save this variant as `explodeShow.html`. The animation is the same except that it is shown in reverse. Like the other effects, `explode` can also make use of specific timings and callback functions.

# The puff effect

Similar to the explode effect but slightly more subtle is the puff effect which causes an element to grow slightly before fading away. Like explode there are few configuration options to concern ourselves with.

Consider a page that has AJAX operations occurring on it. It's useful to provide a loading image that shows the visitor that something is happening. Instead of just hiding an image like this when the operation has completed, we can puff it out of existence instead. Create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/puff.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Puff Effect</title>
  </head>
  <body>
    <img id="loading" alt="Loading" src="img/loading.gif">
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.scale.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //puff the image
        $("#loading").click(function() {
          $(this).hide("puff");
        });
      });
    </script>
  </body>
</html>
```

Save this as `puff.html`. The stylesheet used in this example is purely to position the image slightly so that we can see the full effect of the, well, the effect. For reference, it is comprised of the following styles:

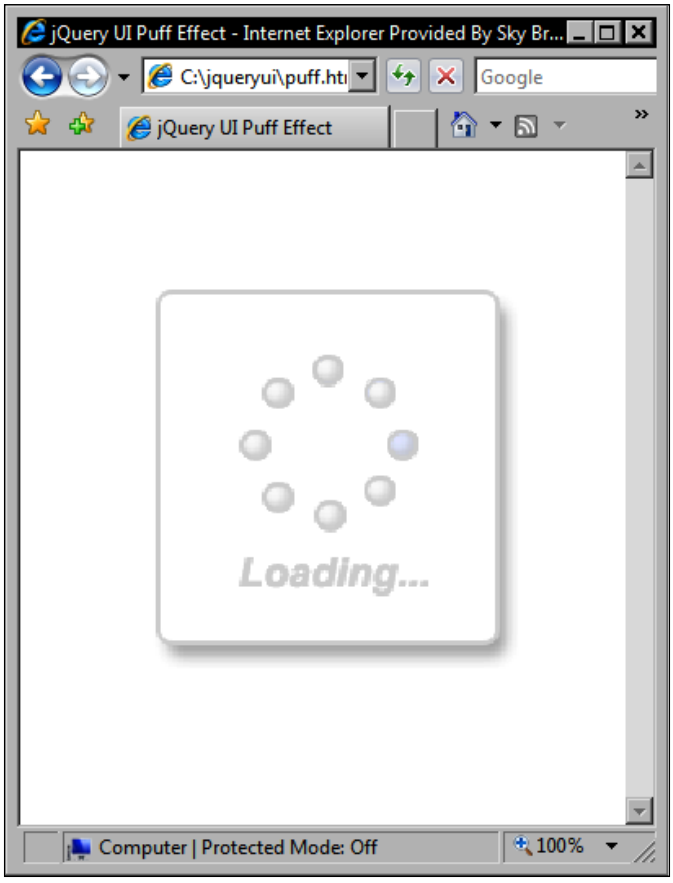
```
#loading { position:relative; top:100px; left:100px; }
```

Save this as `puff.css` in the `styles` folder. In this example, the effect is produced on the click event of the image. However, in reality, we would probably work it into the `success` AJAX event instead.

You'll notice that we used the `effect.scale.js` source file for this effect. The puff effect is the only effect that does not have its own source file and is instead part of the very closely related scaling effect's source file.

Like the explode effect that we looked at in the last section, this effect has just one configuration property that can be passed in an object as the second argument of the `effect` constructor. This is the `percent` property and controls how big the image is scaled up to. The default value is `150%`.

The effect stretches the targeted element (and its children), while at the same time reducing its opacity. It works well on proper images, background colours, and borders, but you should note that it does not work so well with background images specified by CSS. Nevertheless, it's a great effect. The following screenshot shows it in action:



# Pulsate

The pulsate effect is another effect which works with the opacity of a specified element. This effect reduces the opacity temporarily a specified number of times, making the element appear to pulsate.

In the following basic example, we'll create a simple countdown time that counts down from 15. When the display reaches 10 seconds, it will begin to flash red. In a new file, add the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/pulsate.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Pulsate Effect</title>
  </head>
  <body>
    <div id="countdown">15</div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.pulsate.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //set countdown start
        var age = 15;
        //global adjustAge function
        adjustAge = function() {
          //change text of countdown display
          $("#countdown").text(age - 1);
          //start pulsating when 10 is reached
          (age < 11) ? $("#countdown").css({backgroundColor:"#ff0000"}
).effect("pulsate", { times:1 }) : null ;
          //clearInterval when 0 displayed
          (age == 1) ? clearInterval(timer) : age -= 1;
        }
```

```
            //start counting down
            timer = setInterval("adjustAge()", 1000);
          });
        </script>
      </body>
    </html>
```

Save this as `pulsate.html`. Both the page and the script for this example are simple, but the goal is to show off the effect after all. The page itself contains just a simple `<div>` element with the number (as a text string) `15` inside it.

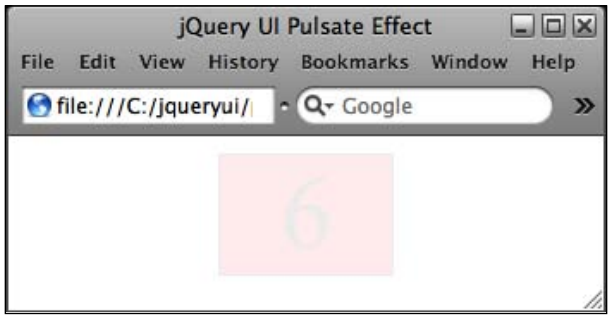
The code first sets a variable equal to the text within the `<div>`. It then defines the global `adjustAge()` function. Unfortunately, this function must be global so that it is visible to the `setInterval` method, which is automatically executed in the context of the `window`.

This function first changes the text content of the specified element to one less than the current `age` variable. It then checks whether `age` has reached `10` yet and if so, applies a background color of red to the element and starts the pulsate effect. It then checks whether the `age` variable has reached `1` yet. If it has, it clears the interval so that it doesn't keep counting down past `0`.

We use the `times` property to specify how many times the element should pulsate. As we'll be executing the method once every second, we can set this to just pulsate once on each call.

After our `adjustAge` function, we then start the interval using JavaScript's `setInterval` function. This function will repetitively execute the specified function after the specified interval, which in this example is `1000` milliseconds, or `1` second.

So every second the number in the countdown `<div>` will decrement by `1` until it gets to `10` when the pulsate effect kicks in. Once the timer reaches `0`, the pulsating stops. The following screenshot shows how the page should appear once the countdown has crossed the 10 second barrier:



# Drop

The drop effect is simple. Elements it is applied to appear to drop off of (or onto) the page, which is simulated by adjusting the element's height and opacity. There are many situations in which this would be useful but one that instantly springs to mind is when creating custom tooltips.

We can easily create a tooltip that appears when an element is hovered over, but instead of just showing the tooltip after a specified period of time has elapsed, we can drop it on to the page instead. Add the following code to a new file in your text editor:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/drop.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Drop Effect</title>
  </head>
  <body>
    <div id="container">
      <p>Lorem <a id="link1" href="#" title="This is a link">ipsum</
a> dolor sit amet, consectetuer adipiscing elit. Sed dapibus
libero non lacus. Morbi <a id="link2" href="#" title="This is
another link">sagittis</a> ante vitae tortor. Quisque quis neque
vel augue laoreet consectetuer. Vestibulum tempor. Morbi non <a
id="link3" href="#" title="This is the third link">justo</a>. Aliquam
ullamcorper, enim sed ultricies accumsan, ipsum mauris eleifend urna,
in ullamcorper nisl urna at erat.</p>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.drop.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //make tooltip on mouseover
        $("#container a").mouseover(function(e) {
          //create tooltip
```

```
        $("<div>").text($(this).attr("title")).addClass("tooltip").
css({left:e.pageX, top:(e.pageY - 40)}).appendTo($("body"));

          //set timeout to show tooltip
          tip = setTimeout("$('.tooltip').show('drop', {
direction:'up' }); ", 750);

            //supress title
            $(this).attr("title", "");

        });

        //make tip track with pointer
        $("#container a").mousemove(function(e) {
          $(".tooltip").css({'left':e.pageX, 'top':e.pageY - 35});
        });

        //remove tooltip on mouseout
        $("#container a").mouseout(function(e) {

            clearTimeout(tip);

          //put title text back
          $("#" + e.target.id).attr("title", $(".tooltip").text());

          //hide and remove tooltip
          $(".tooltip").remove();
          $("#fxWrapper").remove();
        });
      });
    </script>
  </body>
</html>
```

Save this as `drop.html`. The page itself is simple. We've got a container `<div>` and a paragraph with three links inside it. The links are the elements that will trigger our tooltips.

Within our outer document-ready function, we have three distinct anonymous functions. The first is executed when one of the trigger elements fires the `mouseover` event, another is executed on `mousemove`, and the last works with the `mouseout` event.

In the first function, a new `<div>` element is created and its contents are set to the contents of the `title` attribute of the element that fired the `mouseover` event. The new element is given a class of tooltip and has its `left` and `top` style properties set to `35` pixels above the mouse pointer at the time of the event.

Next, a timer is started using JavaScript's `setTimeout` method which will show the new tooltip using the `drop` effect after `750` milliseconds have passed. The `title` attribute of the element that was hovered over is then set to an empty string to prevent the OS default tooltip from appearing.

Our next anonymous function is attached to the `mousemove` event of whichever element fired the initial `mouseover`. Every time the mouse pointer moves our tooltip `<div>` will be repositioned. This means that if the pointer is moved before the tooltip is shown, the tooltip will still appear in the correct location, and while the tooltip is open, it will follow the mouse pointer:

```
//make tip track with pointer
$("#container a").mousemove(function(e) {
  $(".tooltip").css({'left':e.pageX, 'top':e.pageY - 35});
});
```

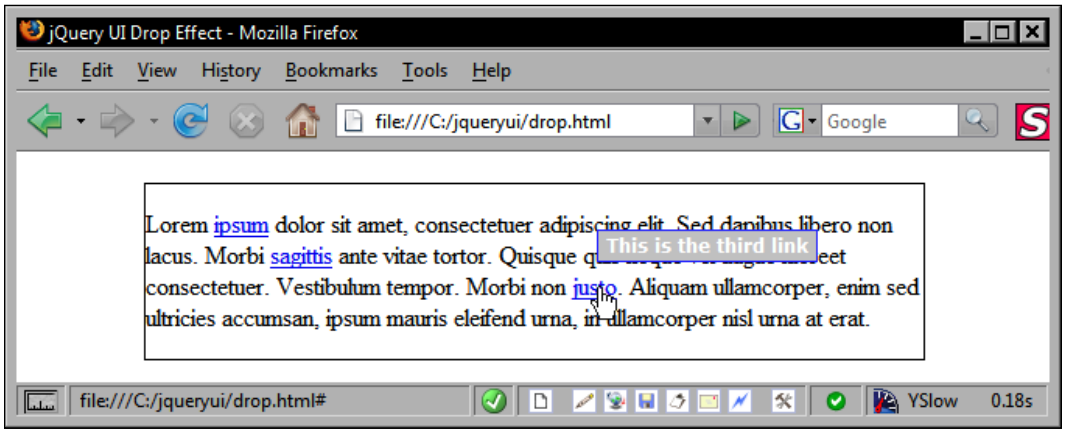
The final function basically tidies up after the tooltip. It clears the timeout (if it is still present) and retrieves the text content of the tooltip to put back on the element's `title` attribute. Finally, it removes the tooltip and the effect wrapper from the DOM, putting everything back as it was:

```
//remove tooltip on mouseout
$("#container a").mouseout(function(e) {
  clearTimeout(tip);
  //put title text back
  $("#" + e.target.id).attr("title", $(".tooltip").text());
  //remove tooltip
  $(".tooltip").remove();
  $("#fxWrapper").remove();
});
```

There is also some minimal CSS required for this example, mostly to style the new tooltip. Create the following stylesheet:

```
#container {
  width:500px; margin:20px auto; border:1px solid #000000;
}
.tooltip {
  background-color:#cccccc; border:1px solid #3333ff;
  color:#ffffff;
  font-family:Verdana; font-weight:bold; font-size:12px;
  position:absolute;
  padding:2px 5px 3px; display:none; z-index:1000;
}
```

Save this in the `styles` folder as `drop.css`. When you run the file in your browser, you should see how the drop effect shows our tooltip, as in the following screenshot:




# Slide

The remaining effects of the jQuery UI library all work by showing and hiding elements in different ways rather than using opacity like most of the effects we have already looked at.

The slide effect is no exception and shows (or hides) an element by sliding it into (or out of) view. It is similar to the drop effect that we just looked. Its main difference is that it does not use opacity.

For our next example, we can create a simple block of text that is initially hidden by an image. The full text can be revealed by sliding the image out of view. In a new page in your text editor, create the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/slide.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Slide Effect</title>
  </head>
  <body>
    <div id="text"><a id="viewAll" href="#" title="View All">view
all</a><span id="ellipse">...</span><p>In Japanese history, a ninja
is an elite warrior, highly trained in all aspects of combat martial
```

```
arts, and specializing in a variety of unorthodox arts of war
</p><p>The methods used by ninja included assassination, espionage,
stealth, camouflage, unconventional warfare, specialized weapons,
and a vast array of martial arts. Their exact origins are still
unknown. Their roles may have included sabotage, espionage, scouting
and assassination missions as a way to destabilize and cause social
chaos in enemy territory or against an opposing ruler, perhaps in the
service of their feudal rulers (daimyo, shogun), or an underground
ninja organization waging guerilla warfare. </p><div id="image">
</div></div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.slide.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //slide div down
        $("#viewAll").click(function() {
            $("#image").toggle("slide", { direction:"down" }, 1000,
function() {
            //change trigger text
            ($("#viewAll").text() == "view all") ? $("#viewAll")
.text("hide").css({color:"#000000"}) : $("#viewAll").text("view all")
.css({color:"#ffffff"}) ;

            //show or hide ellipsis
            ($("#ellipse").css("display") == "block") ? $("#ellipse")
.css({display:"none"}) : $("#ellipse").css({display:"block"}) ;
          });
        });
      });
    </script>
  </body>
</html>
```

Save this as `slide.html`. The page contains a `<div>` element with some other elements nested within it. One of the elements is a `<div>` that hides most of the text with an image, and another is a `<div>` containing three periods to give the impression of an ellipsis. There's also a link used to trigger our effect.

The code is also remarkably simple. When the link is clicked, the image is toggled using the slide effect. If the image is currently being shown, it is hidden and vice-versa. The text of the link is changed to reflect the new state and the ellipsis `<div>` is either shown or hidden.

The default `direction` of the slide effect is set to `left` so we need to use the configuration object to specify that it should be set to `down`. We also supply `1000` milliseconds as the duration of the effect and make use of the callback function to change the link text and ellipsis.

We also use a little CSS in this example. Create the following stylesheet:

```
#text {
  width:400px; border:1px solid #000000; margin:0 auto;
  overflow:hidden; position:relative;
}
#text p { padding:10px; margin:0; }
#viewAll {
  position:absolute; bottom:0px; right:0px;
  z-index:100; color:#ffffff; font-weight:bold;
  text-decoration:none; padding:1px 3px;
}
#ellipsis {
  background-color:ffffff; position:absolute; left:164px;
  top:50px; display:block;
}
#image {
  width:402px; height:208px; background:#000000;
  position:absolute; top:80px;
}
```

Save this as `slide.css` in the `styles` folder. The effect in progress should appear as in the following screenshot:



I said earlier that the `effects.core.js` file had the built-in ability to seamlessly use easing with the effects. Let's see how easy this is to achieve. Change the last `<script>` element in `slide.html` so that it appears as follows (new code shown in bold):

```
//function to execute when doc ready
$(function() {

  //slide div down
  $("#viewAll").click(function() {
    $("#image").toggle("slide", { direction:"down",
easing:($("#viewAll").text() == "view all") ? "easeOutBounce" :
"easeOutBack" }, 1000, function() {
     ($("#viewAll").text() == "view all") ? $("#viewAll").
text("hide").css({color:"#000000"}) : $("#viewAll").text("view all").
css({color:"#ffffff"}) ;
     ($("#ellipsis").css("display") == "block") ? $("#ellipsis").
css({display:"none"}) : $("#ellipsis").css({display:"block"}) ;
    });
  });
});
```

Save this as `slideEasing.html`. See how easy that was. All we need to do is add the `easing` property within our configuration object and define one of the easing methods specified in the easing plug-in by GSGD. Note that we don't actually need to use the separate easing plug-in.

In this example, we specify a different easing method for each toggle state. When the image slides down, it bounce slightly at the end of the animation. When the image slides back up, it will drag the bottom up a little further than it should and then drop it back down.

# Clip

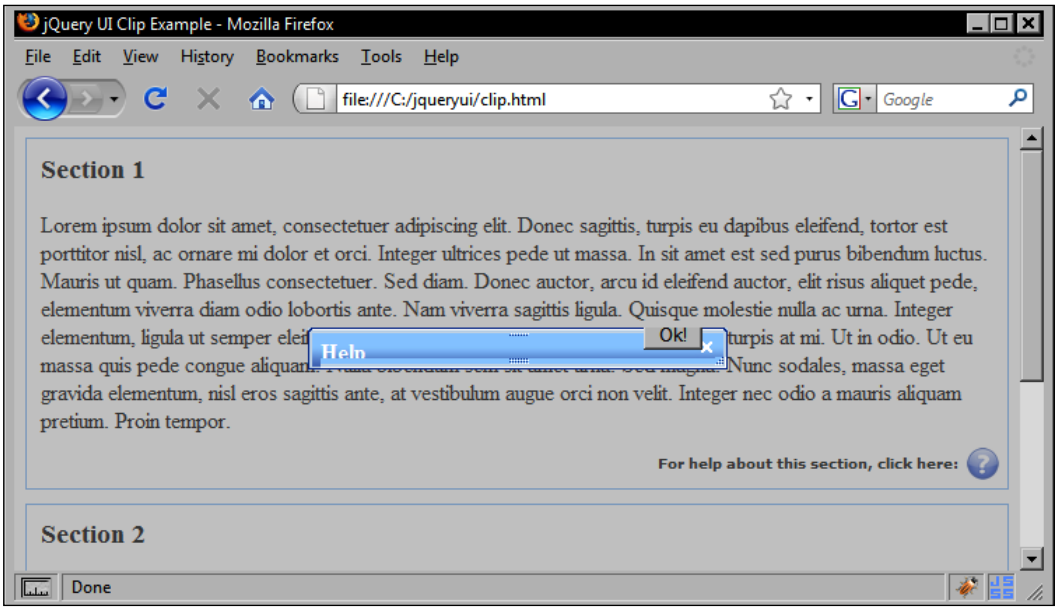
The clip effect is very similar to the slide effect. The main difference is that instead of moving one edge of the targeted element towards the other, to give the effect of the element sliding out of view, the clip effect moves both edges of the targeted element in towards the center.

At the end of chapter 2, we created an AJAX dialog example which showed a small dialog box when one of the help icons was clicked. The dialog could be closed using an **ok** button, which when pressed simply removed the dialog from the page. We could easily use the clip effect to close our dialog instead. In `ajaxDialog.html`, change the `doOk` function so that it appears as follows:

```
//define doOk function
var doOk = function() {
  //close the dialog
  $("#ajaxDialog").parent().parent().hide("clip", {}, "normal",
function() {
    $("#ajaxDialog").dialog("close");
  });
}
```

Save this as `clip.html`. In this simple addition to the existing file, we use the clip effect to hide the dialog from view instead of simply calling the dialog's `close` method. We use the callback built into the `hide` method to call the dialog's `close` method after the effect has finished. This way the dialog still gets closed properly and the modal element gets removed automatically.

When calling the `hide` method, we need to specify all of the arguments, but as we are happy with the default configuration and the normal speed, we simply supply an empty object as the second argument and the string `normal` as the third argument. The next screenshot shows the dialog being clipped:



The clip effect also has just a single native configuration property. This is the `direction` property that we already saw in the drop and slide effects, but this time the property may take just one of two values instead of four. The values that the clip effect's `direction` property accepts are `horizontal` or `vertical`, with `vertical` being the default.

# Blind

The blind effect is practically the same as the slide effect that we looked at earlier. Visually, the element appears to do the same thing, and the two effects' code files are also extremely similar. The main difference between the two effects that we need to worry about is that with this effect we can only specify the axis of the effect, not the actual direction.

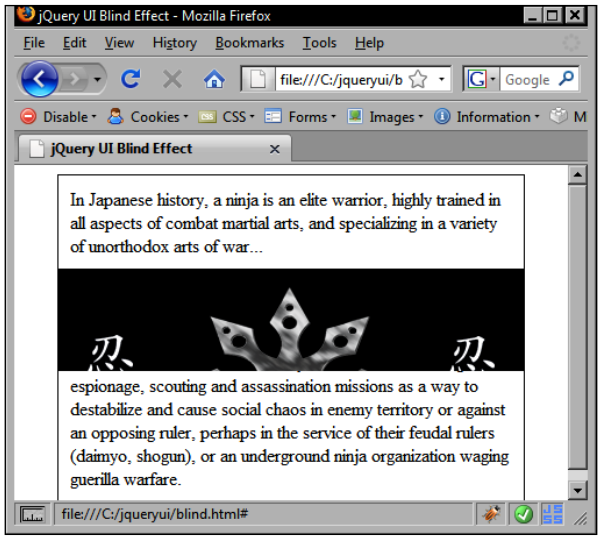
Like the clip effect that we looked at in the last section, the `direction` property that this effect uses for configuration only accepts the values `horizontal` or `vertical`. For example, when we used the slide effect earlier, we were able to slide the top of a `<div>` element down to the bottom to reveal the text. If we used the blind effect instead, the bottom of the `<div>` would slide up to the top to reveal the text. Try it out. Change `slide.html` to the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/slide.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Blind Effect</title>
  </head>
  <body>
    <div id="text"><a id="viewAll" href="#" title="View All">view
all</a><span id="ellipsis">...</span><p>In Japanese history, a ninja
is an elite warrior, highly trained in all aspects of combat martial
arts, and specializing in a variety of unorthodox arts of war
</p><p>The methods used by ninja included assassination, espionage,
stealth, camouflage, unconventional warfare, specialized weapons,
and a vast array of martial arts. Their exact origins are still
unknown. Their roles may have included sabotage, espionage, scouting
and assassination missions as a way to destabilize and cause social
chaos in enemy territory or against an opposing ruler, perhaps in the
service of their feudal rulers (daimyo, shogun), or an underground
ninja organization waging guerilla warfare.</p><div id="image"></
div></div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
```

```
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.blind.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //blind div up
        $("#viewAll").click(function() {
            $("#image").toggle("blind", { direction:"vertical" },
1000, function() {
            ($("#viewAll").text() == "view all") ? $("#viewAll")
.text("hide").css({color:"#000000"}) : $("#viewAll").text("view all")
.css({color:"#ffffff"}) ;
            ($("#ellipsis").css("display") == "block") ?
$("#ellipsis").css({display:"none"}) : $("#ellipsis")
.css({display:"block"}) ;
          });
        });
      });
    </script>
  </body>
</html>
```

Save this as `blind.html`. Literally, all we've changed is the string specifying the effect, in this case to `blind`, and the value of the `direction` property from `down` to `vertical`. When you run the file however, you should notice the difference instantly.

The effect does indeed look very much like your typical window blind, either rolling up or rolling back down:



# Fold

Folding is a neat effect that gives the appearance that the element it's applied to is being folded up like a piece of paper. It achieves this by moving the bottom edge of the specified element up to `15` pixels from the top, then moving the right edge completely over towards the left edge.

The distance from the top that the element is shrunk to in the first part of this effect is exposed as a configurable property by the API. So, this is something that we can adjust to suit the needs of our implementation. This property is an integer. The best way to judge the effect for ourselves is to put it to work. Create the following page:

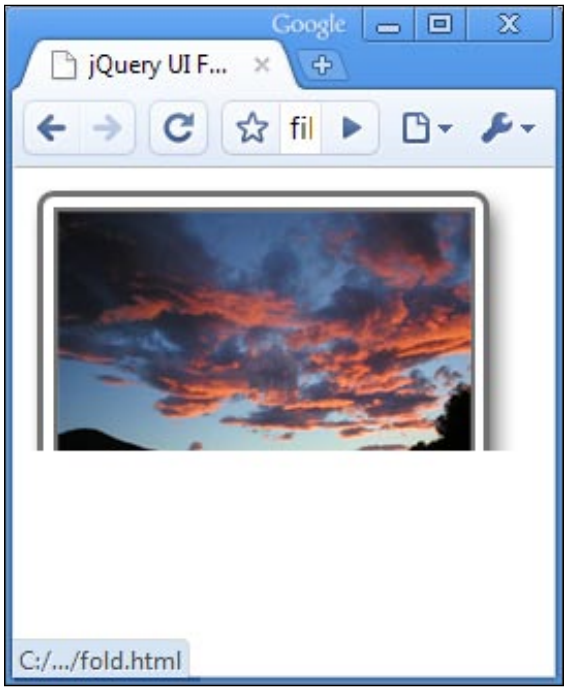
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="styles/fold.css">
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>jQuery UI Fold Effect</title>
  </head>
  <body>
    <div class="container">
      <div class="image"><img src="img/sunset1.jpg" alt="Sunset">
</div>
      <div class="ui"><a class="close" href="#"></a></div>
    </div>
    <script type="text/javascript" src="jqueryui1.6rc2/
jquery-1.2.6.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.core.js"></script>
    <script type="text/javascript" src="jqueryui1.6rc2/ui/
effects.fold.js"></script>
    <script type="text/javascript">
      //function to execute when doc ready
      $(function() {
        //fold image when close clicked
        $(".close").click(function() {
          $(this).parent().parent().hide("fold", { size:"50" });
        });
      });
    </script>
  </body>
</html>
```

Save this as `fold.html`. The page features a single image, with a simple placeholder and a close button. When the button is clicked, our anonymous function traverses up the DOM to its grandparent and then applies the fold effect to it.

We make use of the `size` configuration property to make the effect stop the first fold `50` pixels before the top of the element. We also need some CSS for this example. Create the following stylesheet:

```
.container {
  width:234px; height:211px;
  position:relative;
  background:url(../img/placeholder.png) no-repeat;
}
.image {
  position:relative; top:11px; left:11px;
  margin-bottom:10px;
}
.ui {
  width:200px;
  position:relative; left:11px;
  border:2px solid #666666;
}
.close {
  width:60px; height:20px;
  font-family:verdana; font-size:80%; font-weight:bold;
  text-align:center; text-decoration:none;
  background:url(../img/close.gif) no-repeat;
  position:relative; left:135px; top:2px;
  display:block;
}
.close:hover {
  background:url(../img/close_over.gif) no-repeat;
}
img { border:2px solid #666666; }
```

Save this file as `fold.css` in the `styles` folder. The effect in progress should appear as in the following screenshot:




# Summary

In this chapter, we've covered the complete range of UI effects available in the jQuery UI library. We've seen how easy it is to use the `effects.core.js` base component to construct attractive color animations and smooth class transitions.

We also saw that the following effects can be used in conjunction with the simple effect API:

- bounce
- highlight
- shake
- transfer

An important point is that most of the individual effects can be used not only with the effect API but can also make use of `show/hide` and `toggle` logic, making them incredibly flexible and robust. The following effects can be used with this advanced API:

- blind
- clip
- drop
- explode
- fold
- puff
- pulsate
- scale
- slide

This now brings us to not only the end of this chapter, but also the end of the book. There is a saying, I'm sure that you've all heard it before; it's the "give a man a fish..." saying. I hope that during the course of this book, I've taught you how to fish instead of just giving you a fish.

The aim of the examples that we've worked through over the chapters has not been just to show you how to use the library but also to show you that it is powerful enough and flexible enough to be limited only by your imagination. The world class interfaces of tomorrow are made possible today with jQuery UI.

# Index


## A

**accordion methodology**
about 72
accordion widget, removing 72, 73
activate method 78, 79
activate method, testing 78
destroy method 72, 73
disable method 74-77
enable method 74-77
**accordion widget**
about 57
accordion methodology 72
animation 79
configurable properties 65, 66
configurable properties, using 66-68
configuring 65
custom styling 61-65
elements 58
events 81, 83
implementing 58, 59
jQuery UI accordion navigation example 83-87
navigation menu, building 83
script file, required 60
structure 58
**accordion widget, styling**
custom stylesheet, creating 61
custom theme, creating 61
Firebug plugin, used 61
flora theme, used 61
**AJAX**
date picker widget, modifying 175
**AJAX tabs, UI tabs widget**
about 46
creating 46-50
**animation, accordion widget**
easing methods 80
slide animation 79
**auto complete widget**
about 183
additional data, sending to server 210
advanced formatting 198-205
basic implementation 184, 185
caching 210
configurable properties 186, 187
email field example with JSON 214-218
matching properties 205
methods 211
multiple sections 197
remote data 207, 208
styling 185-197

## B

**browser support, jQuery UI**
Chrome 19
Firefox 2 19
Firefox 3 19
IE 6 19
IE 7 19
IE 8 19
Opera 9 19
Safari 3 19

## C

**callback properties, date picker widget**
about 156
beforeShow 156
beforeShowDay 156
calculateWeek 156
onChangeMonthYear 156

onClose 156
onSelect 156
statusForDate 156
using 157, 158
**callback properties, dialog widget**
about 100
close 100
drag 100
dragStart 100
dragStop 100
focus 100
open 100
resize 100
resizeStart 100
resizeStop 100
uses 101, 102
**callback properties, draggables**
about 238
absolutePosition property 238
drag property 238
functions 238-242
helper property 238
options property 238
position property 238
start property 238
stop property 238
**callback properties, droppables**
about 254
activate property 254
deactivate property 254
drop property 254
functions 254-257
out property 254
over property 254
**callback properties, resizable**
resize 286
start 286
stop 286
**callbacks properties, selectables**
selected 301
selecting 301
start 301
stop 301
unselected 301
unselecting 301
working 301-304
**component categories, jQuery UI**
higher-level widgets 18
low-level interaction helpers 18
**configurable properties, accordion widget**
active property 69
alwaysOpen property 70
animated property 79
autoHeight property 70
clearStyle property 70
event property 67
fillSpace property 70
navigation property 70
**configurable properties, auto complete widget**
autoFill property 188
formatItem property 199, 200
formatMatch property 204
formatResult property 203, 205
highlight property 194
minChars property 190
multiple property 197
multipleSeparator property 198
mustMatch property 205
scrollHeight property 190
scroll property 190
selectFirst property 189
width property 193
**configurable properties, date picker widget**
about 144
appendText property 148
buttonImage property 160
callback properties 156
changeFirstDay property 148
changeMonth property 148
changeYear property 148
closeAtTop property 148
dateFormat property 148, 150
duration property 148
numberOfMonths property 162
rangeSelect property 164
regionalization properties 150, 151, 152, 154
showAnim property 164
showOn property 159
showOption property 164
showOtherMonths property 148
showWeeks property 148

**configurable properties, selectables**
autofresh 298
filter 298
**connected sortables 347**
**core effects file**
about 376
advanced easing 380
class transitions 378, 380
color animations 376
color animations, implemeting 376, 377
style attributes, color animations 378

## D

**date picker widget**
about 139
AJAX magic 176
alternative animations, configuring 164, 165
configurable properties 144-148
default date picker, creating 140, 141
elements 139
internationalizing 153-155
localizing 150
methods 166
modifying 175-181
multiple months, implementing 161, 162
putting in a dialog 168-174
range selection, enabling 163, 164
skinning 142, 143
trigger buttons 159, 160
**dialog widget**
about 89
AJAX dialog, creating 111-115
animations 102, 103
basic dialog, creating 90, 91
callback properties 100
custom skins 92
data, getting from 108-111
elements 89
jQuery UI AJAX dialog example 111-116
methods 104, 105
properties 94
skinning 92-94
**drag and drop example 261-267**
**draggables**
about 219-221
basic implementation 221, 222
callback functions 238
drag, constraining 233-235
dragged elements, resetting 227, 228
handles, dragging 228, 229
helper elements, dragging 230-233
methods 243
properties 223, 224
properties, configuring 223
properties, using 224-227
snapping, configuring 236, 237
**draggables API 219**
**droppables**
about 219, 220, 244
callback properties 254
default implementation 245, 246
methods 261
properties, configuring 247
**droppables API 219**

## E

**encapsulation 222**
**event handlers, sortables**
about 341
activate 341
beforeStop 341
change 341
deactivate 341
functions 342-346
out 341
over 341
receive 341
remove 341
sort 342
start 342
stop 342
update 342

## F

**Fun with UI widgets**
accordion 83
auto-complete 214
date picker 175
dialog 111
droppables 261
resizables 289

selectables 308
sortables 356
tabs 52

## G

**greed property, droppables**
about 257
example 258-260

## H

**helpers, sortables 334**

## I

**implementation, UI tabs widget**
<href>element, HTML elements 25
HTML elements, used 24
list element, HTML elements 25

## J

**jQuery UI**
about 9
book examples 20
browser support 19
component categories 18
draggables 219
droppables 219
library licensing 21
simplified API 17
theme roller 16, 17
ui.core.js file 18
UI effects 375
**jqueryui1.6rc2 folder, jQuery UI library structure**
_MACOSX directory 13
demo directory 13
qunit folder, test directory 14
test folder 14
theme directory 14
ui folder 15
**jQuery UI library**
accordion widget 57
auto complete widget 183
date picker widget 139
development environment, setting up 12, 13
dialog widget 89
downloading 11
selectables 293
slider widget 117
sortables 321
structure 13
UI tabs widget 23
**jQuery UI library licensing**
GPL lincense 21
MIT lincense 21

## L

**library files**
jquery-1.2.6.js 185
ui.all.css 185
ui.autocomplete.js 186
ui.core.js 185
**library files, draggables**
jquery-1.2.6.js 222
ui.core.js 222
ui.draggable.js 222
**library files, droppables**
jquery-1.2.6.js 247
ui.core.js 247
ui.draggable.js 247
ui.droppable.js 247
**library files, sortables**
jquery-1.2.6.js 323
ui.core.js 323
ui.sortable.js 323

## M

**methods, auto complete widget**
destroy 211
flushCache 211, 213
result 211
search 211
setData 211, 213
**methods, date picker widget**
change 166, 167
destroy 166
dialog 166, 168
disable 166

enable 166
getDate 166
hide 166
isDisabled 166
setDate 166
show 166
**methods, dialog widget**
close 104
destroy 104
isOpen 104
moveToTop 104
open 104
uses 105-108
**methods, draggables**
destroy method XE 243
enable method 243
functions 243, 244
toggle() function, calling 244
**methods, droppables**
destroy method 261
disable method 261
function 261
**methods, slider widget**
moveTo 128
value 128
**methods, sortables**
destroy method 351
disable method 351
enable method 351
functions 351-354
in action 351
properties, used 354
refresh method 351
refreshPositions method 351
serialize method 351
toArray method 351
**methods, UI tabs widget**
add method 37, 39
destroy method 37, 45
disable method 37
enable method 37
length method 37, 43
load method 37
remove method 37, 39
rotate method 37, 43
select method 37, 42
url method 37

# P

**placeholders**
about 331
**properties, dialog widget**
autoOpen property 95
button property 98, 99
height property 99
modal property 97
overlay property 97
position property 96
title property 96
**properties, draggables**
axis property 227
clone property 231
container property 233
containment property 235
cursorAt property 226
cursor property 224
delay property 227
distance property 226
grid property 227
handle property 228
helper property 231
left property 226
opacity property 233
revert property 231
scroll property 235
snapMode property 236
snap property 236
snapTolerance property 236
steps property 227
**properties, droppables**
accept property 247, 248
activeClass property 247
greedy property 247
hoverClass property 247, 248
modes, tolerance property 251
tolerance property 247, 251
uses 247-250
**properties, resizable**
all property 278
animateDuration property 284
animateEasing property 284
animate property 284
autoHide property 278
containment property 283

ghost property 281
handles property 275
helper property 285
knobHandles property 278
**properties, sortables**
configuring 325, 327
connectWith property 338-341
containment property 328
cursor property 328
delay property 330
distance property 328
forcePlaceholderSize property 332
functions 327-330
handle property 330
helper property 334
items property 336, 337
opacity property 330
placeholder property 331, 333
revert property 330
**properties used with remote data, auto complete widget**
cacheLength 207
extraParams 207
matchCase 207
matchSubset 207
url 207

## R

**regionalization properties, date picker widget**
clearStatus 150
clearText 150
closeStatus 150
closeText 150
currentStatus 151
currentText 151
dateFormat 151
dateNames 151
dateNamesMin 151
dateNamesShort 151
dateStatus 151
dayStatus 151
firstDay 151
iniStatus 151
monthNames 151
monthNamesShort 151
monthStatus 151
nextStatus 151
nextText 151
prevStatus 151
prevText 151
weekHeader 151
weekStatus 151
yearStatus 151
**resizable**
about 269, 270
animations 284, 285
basic resizable, implementing 270-272
callback properties 286, 288
default flora theme 272
ghost elements, resizing 280, 281
jQuery UI resizable tabs example 289-292
methods 289
properties 274, 275
resized element ratio, maintaining 282
resized elements, constraining 282, 283
resize handles, configuring 275-278
size limits, defining 279, 280
skinning 273, 274

## S

**script files, accordion widget**
accordion source file 60
jQuery library 60
UI base file 60
**selectable class**
about 297
configurable properties 298
**selectable methods**
about 304
destroy 304
disable 304
enable 304
refresh 304
toggle 304
**selectables**
about 293
basic image viewer, creating 308-319
basic implementation 294
callbacks properties 301
default implementation, creating 294-296
filtering 299, 300

jQuery UI selection example 308-319
methods 304-308
selectee class names 297
**selectee class names 297**
**slider widget**
about 117
animation 131
appearance, changing 119
callback functions, using 125-127
color slider example 134-137
configurable properties 122
creating 118, 119
default theme, overriding 119-121
elements 117
implementing 118, 119
methods 127
moveTo method, using 128-130
multiple handles 131
multiple handles, implementing 131-134
slider background, elements 117
slider handle, elements 117
stepping property 122-125
steps property 122-125
vertical slider, creating 121
**sortables**
about 321
basic implementation 321-324
callback properties 341
connected events 347
connected events, in action 348-350
connected lists 338-341
event handlers 341
helpers 334, 335
items 336
JavaScript, jQuery UI customisable home page example 360-371
jQuery UI customisable home page example 356-360
library files 323
methods 351
placeholders 331
properties 325-327
properties, configuring 325
widget compatibility 354, 356
**sortables helpers 334**
**sortables items 336**
**structure, jQuery UI library**
i18n folder 15
jqueryui1.6rc2 folder 13
minified components 15
packed components 15
unit testing 14
widget theming 15

## T

**theme roller**
about 16, 17
preview 17
**tolerance property, droppables**
about 251
fit mode 251
intersect mode 251
pointer mode 251, 252
touch mode 251, 253

## U

**UI effects**
about 375
additional effect parameters, highlight effect 382, 383
blind effect 407, 408
bounce effect 384, 385
clip effect 405, 406
core effects file 376
drop effect 399-401
explosion effect 392-394
fold effect 409, 410, 411
highlight effect 381, 382
properties, bounce effect 384
properties, scale effect 392
properties, shake effect 386
properties, transfer effect 390
puff effect 395, 396
pulsate effect 397, 398
scale effect 390
shake effect 385, 386
slide effect 402-405
transfer effect 387-389
**UI tabs widget**
about 23
AJAX tabs 46

components 23
configured properties, using 29, 30
custom events 34
event handler, binding with custom event 34
in conjunction with, jQuery library getJSON method 52, 54
jQuerybind() mehod 34
methods 37
properties 33
tab, implementing 24
tab, styling 26, 27
tab carousel, creating 43-45
tab events 33
tab implementation 24
tab implementation, underlying HTML elements used 24
tab implementation example 24
tab properties, configuring 28, 29
tabs, adding 39-42
tabs, configuring 28
tabs, disabling 37, 38
tabs, enabling 37, 38
tabs, removing 39-42
tabs methods, using 37
transition effects, enabling 31-33



Thank you for buying
# jQuery UI 1.6: The User Interface Library for jQuery

## Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing jQuery UI 1.6: The User Interface Library for jQuery, Packt will have given some of the money received to the jQuery project.

In the long term, we see ourselves and you—customers and readers of our books—as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

## Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

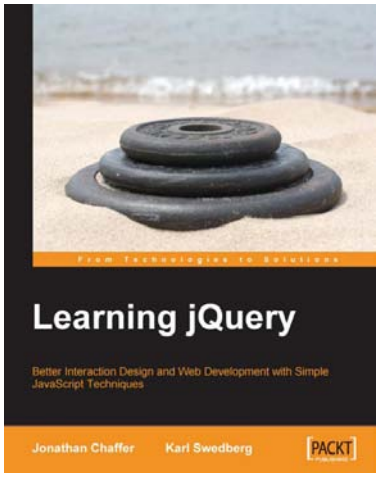
## About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.PacktPub.com`.

PACKT PUBLISHING




## Learning jQuery

ISBN: 978-1-847192-50-9 Paperback: 380 pages

Better Interaction Design and Web Development with Simple JavaScript Techniques

1. Create better, cross-platform JavaScript code
2. Learn detailed solutions to specific client-side problems
3. For web designers who want to create interactive elements for their designs
4. For developers who want to create the best user interface for their web applications.

---




## Object-Oriented JavaScript

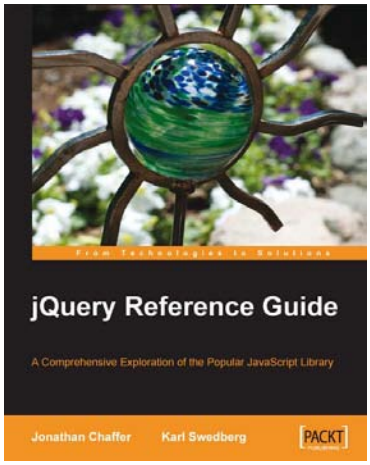
ISBN: 978-1-847194-14-5 Paperback: 337 pages

Create scalable, reusable high-quality JavaScript applications and libraries

1. Learn to think in JavaScript, the language of the web browser
2. Object-oriented programming made accessible and understandable to web developers
3. Do it yourself: experiment with examples that can be used in your own scripts
4. Write better, more maintainable JavaScript code

Please check **www.PacktPub.com** for information on our titles

PACKT PUBLISHING



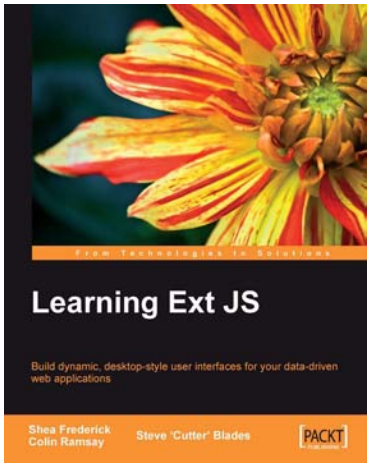

## jQuery Reference Guide

ISBN: 978-1-847193-81-0 Paperback: 225 pages

A Comprehensive Exploration of the Popular JavaScript Library

1. Organized menu to every method, function, and selector in the jQuery library
2. Quickly look up features of the jQuery library
3. Understand the anatomy of a jQuery script
4. Extend jQuery's built-in capabilities with plug-ins, and even write your own

---




## Learning Ext JS

ISBN: 978-1-847195-14-2 Paperback: 299 pages

Build dynamic, desktop-style user interfaces for your data-driven web applications

1. Learn to build consistent, attractive web interfaces with the framework components
2. Integrate your existing data and web services with Ext JS data support
3. Enhance your JavaScript skills by using Ext's DOM and AJAX helpers
4. Extend Ext JS through custom components

Please check **www.PacktPub.com** for information on our titles