



Quick answers to common problems

jQuery 2.0 Development Cookbook

Over 80 recipes providing modern solutions to web development problems with real-world examples

Leon Revill

[PACKT] open source 
PUBLISHING community experience distilled

jQuery 2.0 Development Cookbook

Over 80 recipes providing modern solutions to web development problems with real-world examples

Leon Revill

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

jQuery 2.0 Development Cookbook

Copyright © 2014 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, nor Packt Publishing, and its dealers and 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 of 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: February 2014

Production Reference: 1140214

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-089-6

www.packtpub.com

Cover Image by Karl Moore (karl.moore@ukonline.co.uk)

Credits

Author

Leon Revill

Project Coordinator

Joel Goveya

Reviewers

Gary Gilbert

Joni Halabi

Proofreaders

Maria Gould

Ameesha Green

Paul Hindle

Acquisition Editors

Akram Hussain

Sam Wood

Indexer

Rekha Nair

Content Development Editor

Arvind Koul

Production Coordinator

Alwin Roy

Technical Editors

Dennis John

Pankaj Kadam

Gaurav Thingalaya

Cover Work

Alwin Roy

Copy Editors

Tanvi Gaitonde

Dipti Kapadia

Aditya Nair

About the Author

Leon Revill has over five years' commercial web development experience with PHP5 and MySQL technologies on large and small projects. His development skillset extends over many JavaScript technologies, such as jQuery, AngularJS, and NodeJS. Being an all-round tech enthusiast, some of Leon's spare time is spent working on personal projects to allow him get to grips with new technologies quickly. Leon runs a web development blog (<http://www.revillweb.com/>), where he shares his knowledge in the form of articles and tutorials.

I would like to thank my friends and family, who have been incredibly supportive while writing this book. A special thanks goes to Allýce Wolverson, whose support during long days and difficult times has given me the opportunity to complete this title.

Finally, I would like to thank everyone at Packt Publishing for all their hard work in making this book possible.

About the Reviewers

Gary Gilbert first got his start with programming, hacking on Ti 99/4a and VIC-20 systems in his early teens in Canada, and was instantly hooked. After graduating from University, Gary moved to England to work for an IT consultancy, where he developed desktop applications. Two years and seven addresses later, Gary found himself working for a government contractor in Washington DC, building web applications using a variety of client/server technologies, such as HTML and pure JavaScript. In early 2007, Gary was introduced to jQuery 1.1 and in 2010, he began developing web applications with jQuery full time.

Gary Gilbert is currently Deputy Manager of software development at CONTENTS Software GmbH in Munich, Germany, where he helps develop the company's next-generation content management software.

I would like to personally thank the jQuery team for their dedication and hard work in developing the library; their tireless efforts have made my work much, much easier. I would also like to thank Packt Publishing for giving me the opportunity to review this book.

Joni Halabi is a Senior User Experience Developer at Optaros with over 10 years' experience in frontend website development. In this role, she has worked with a wide variety of clients to provide them with robust and user-focused solutions.

Joni's technical expertise includes the frontend development of complex website designs on a variety of popular frameworks, including Magento, Hybris, Drupal, and WordPress. Her talents also include JavaScript and jQuery development, as well as code optimization to meet cross-browser and mobile browser requirements.

Prior to Optaros, Joni managed and developed with a team of talented U/I engineers for a Cambridge-based online gaming company and taught web development and graphic design at several elementary schools in upstate New York. Joni is also a certified Magento frontend developer.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print and bookmark content
- ▶ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Document Object Model Manipulation	5
Introduction	5
Selecting elements	6
Finding and selecting sibling elements	9
Creating DOM elements	11
Inserting content into an element	14
Modifying the DOM element properties	17
Adding and removing CSS classes to dynamically change their style	18
Enabling and disabling buttons by changing their properties	21
Updating an image within a page	24
Populating list elements	27
Understanding pagination	29
Removing DOM elements	36
Re-using DOM elements	38
Chapter 2: Interacting with the User by Making Use of jQuery Events	43
Introduction	43
Detecting button clicks	44
Detecting element clicks	46
Detecting change	48
Updating content based on user input	51
Detecting key press events on inputs	53
Restricting input character length	56
Changing page elements on mouse hover	58
Triggering events manually	61
Preventing event triggers	63
Creating a custom event	66

Chapter 3: Loading and Manipulating Dynamic Content with AJAX and JSON	69
Introduction	70
Loading HTML from a web server into a page	70
Using AJAX and handling server errors	76
Processing JSON data	79
Searching JavaScript objects	83
Sorting JavaScript objects	87
Caching JSON and AJAX requests	90
Creating a search feature	93
Creating an autosuggest feature	106
Waiting for an AJAX response	114
Chapter 4: Adding Attractive Visuals with jQuery Effects	119
Introduction	119
Sliding page elements	120
Hiding and showing elements	124
Fading elements	126
Toggling effects	130
Stopping effects	133
Chaining effects	136
Creating a basic photo gallery	138
Creating a blinking button	148
Removing elements with effects	151
Chapter 5: Form Handling	155
Introduction	155
Implementing basic form validation	156
Adding number validation	164
Adding credit card number validation	168
Adding date validation	170
Adding e-mail address validation	173
Implementing live form validation	175
Adding a password strength indicator	177
Adding anti-spam measures	184
Implementing input character restrictions	187
Chapter 6: User Interface	191
Introduction	191
Manipulating element CSS	192
Creating a news ticker	197
Creating sticky elements	201

Implementing smooth scrolling	206
Creating a dynamic table of contents	210
Creating a basic drag-and-drop functionality	215
Creating a dynamic animated tree menu	221
Creating an accordion content slider	226
Creating tabbed content	232
Creating a modal pop up	236
Creating a draggable content pop up	240
Chapter 7: User Interface Animation	245
Introduction	245
Creating an animated login form	245
Adding photo zoom	253
Creating an animated content slider	258
Animating background images	263
Creating an animated navigation menu	267
Chapter 8: Understanding Plugin Development	275
Introduction	275
Creating a plugin template	276
Creating a tooltip plugin	277
Building a content and image slider plugin	281
Creating an RSS feed reader plugin	286
Coding an image cropper plugin from scratch	292
Chapter 9: jQuery UI	311
Introduction	311
Creating stylish and functional buttons	312
Creating dialog boxes for user information and input	315
Implementing progress bars within your application	319
Adding date picker interfaces to input boxes quickly	323
Creating an autocomplete search feature	327
Chapter 10: Working with jQuery Mobile	335
Introduction	335
Creating a basic mobile website template	336
Building a complete static website	338
Building a dynamic mobile website	341
Implementing the quick call functionality	348
Implementing the send SMS functionality	349
Adding mobile-friendly lists	351
Using touch-oriented events	355

Table of Contents

Creating mobile-compatible forms	358
Building a complete registration and login system	363
Building a complete mobile web app	376
Index	389

Preface

jQuery 2.0 Development Cookbook will provide you with many reusable code recipes to create common and unique website and web application elements, plugins, and interfaces using the most popular client-side framework, jQuery. Following the step-by-step instructions for each of the recipes will not only provide you with useable code, but also the understanding needed to extend and improve on it.

What this book covers

Chapter 1, Document Object Model Manipulation, covers how to use jQuery to manipulate the 186 web page's HTML code on the client to create a rich and visual user experience.

Chapter 2, Interacting with the User by Making Use of jQuery Events, harnesses the power of jQuery to detect and respond to user interactions, which creates intuitive user interfaces.

Chapter 3, Loading and Manipulating Dynamic Content with AJAX and JSON, utilizes jQuery's AJAX functionality with JSON-formatted data to bring pages to life by updating content without the need for a page refresh.

Chapter 4, Adding Attractive Visuals with jQuery Effects, explains how to add shine to your website or web application with jQuery's effects and basic animations to create unforgettable designs.

Chapter 5, Form Handling, covers how to use jQuery to build robust client-side validation and an intuitive user experience for web forms.

Chapter 6, User Interface, covers how to break the mold and create powerfully intuitive interfaces from scratch and engage the user with a high level of interactivity.

Chapter 7, User Interface Animation, covers how to extend upon jQuery's built-in animation and combine CSS with jQuery to create fabulous website modules for use with any website.

Chapter 8, Understanding Plugin Development, explains how to create reusable code that provides solutions to a range of common website and web application problems.

Chapter 9, jQuery UI, covers how to empower your website or web application with jQuery's user interface library to create attractive and user-friendly page elements and interfaces.

Chapter 10, Working with jQuery Mobile, covers how to create a mobile and cross-platform-ready website using jQuery's powerful mobile framework.

What you need for this book

For all the recipes in this book, you will require an IDE to write JavaScript, HTML, and CSS code, and a web browser to execute your code. For some of the more advanced recipes in this book, you will require a web server running MySQL and PHP.

Who this book is for

This book is for anyone who is either new to jQuery and looking to learn some basics, or familiar with jQuery and looking to expand their knowledge and create some advanced components for their website or web application. This book is an excellent resource for web developers of all skill and experience levels.

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, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Any code within `$(function() { }) ;` will be automatically executed by jQuery when the page is loaded."

A block of code is set as follows:

```
<!DOCTYPE html>
<html>
<head>
  <title>Creating DOM elements</title>
  <script src="jquery.min.js"></script>
  <script></script>
</head>
<body>
  <div id="container">
    <ul id="myList">
      <li>List Item 1</li>
```

```
<li>List Item 2</li>
<li>List Item 3</li>
</ul>
</div>
</body>
</html>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "This will display a pop-up window to the user that has the message **Are you sure you want to delete this user?**"



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 send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

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

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from https://www.packtpub.com/sites/default/files/downloads/0896OS_GraphicsBundle.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any 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 us with the location address or website name immediately so that 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 any aspect of the book, and we will do our best to address it.

1

Document Object Model Manipulation

In this chapter, we will cover:

- ▶ Selecting elements
- ▶ Finding and selecting sibling elements
- ▶ Creating DOM elements
- ▶ Inserting content into an element
- ▶ Modifying the DOM element properties
- ▶ Adding and removing CSS classes to dynamically change their style
- ▶ Enabling and disabling buttons by changing their properties
- ▶ Updating an image within a page
- ▶ Populating list elements
- ▶ Understanding pagination
- ▶ Removing DOM elements
- ▶ Re-using DOM elements

Introduction

This chapter looks at the fundamental principles of jQuery—finding, selecting, and manipulating DOM elements. jQuery makes it easy for JavaScript developers to select single or multiple HTML page elements using a variety of methods.

Once the developer has selected these elements, jQuery provides the ability to manipulate each of these elements in order to create a richer user experience through attribute modifications such as style, disabled, and class.

Selecting elements

There are many ways in which you can use jQuery to select DOM elements. We will explore the main methods here. For developers familiar with CSS, it is possible to use the same syntax when selecting elements with jQuery (that is, `#content`, `.content`, and so on).

Getting ready

Open a blank HTML document within your text editor or IDE of choice. Ensure that you have the latest version of jQuery downloaded and is easily accessible for inclusion into this HTML document. When creating new HTML files within this chapter, ensure that they are all within the same directory as the jQuery library file, making it easy to include into the HTML document.

How to do it...

To understand how you can use jQuery to select a variety of DOM elements, perform each of the following recipe steps:

1. Create a web page using the following HTML and JavaScript code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Selecting Elements with jQuery</title>
  <script src="jquery.min.js"></script>
  <script>
    $(function() {
      var content = $("#content"); //Select the content
      div
      var span = $(".span-element"); //Select the span
      element
      var listelements = $("li"); //Select all the list
      elements
    });
  </script>
</head>
<body>
  <div class="division-container">Some text within a div
    which has a class</div>
```

```

<div id="content">Some text within a div which has an ID
  attribute</div>
<a href="#">A link</a>
<a href="#" rel="dofollow">A second link</a>
<ul class="info-list">
  <li>List Item 1</li>
  <li>List Item 2</li>
  <li>List Item 3</li>
</ul>
<button>Button 1</button>
<span class="span-element">Span 1</span>
</body>
</html>

```

2. To select any of these elements, use the jQuery's `$()` function. We can use this function in conjunction with an identifier or CSS selector for an element we would like to select; for example, its HTML tag `li` and ID `#content` or a class `.content`.



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

How it works...

The simplest method of selecting a DOM element is by its ID. We know that all IDs within a HTML document should be unique; therefore, by selecting an element with its ID, you will be selecting a single element.

In reference to the preceding HTML document, if you wanted to select `<div>`, which has an ID `content`, you can use the following jQuery code to select it:

```

$(function() {
  var content = $('#content');
});

```

This would make the DOM element available within the `content` variable. More on what this means is covered later in the chapter.



Any code within `$(function() { });` will be automatically executed by jQuery when the page is loaded.

We can also select elements in the same way using their class. The code is very similar to the preceding example, except that we use the class prefix (.) instead of the ID prefix (#), illustrated as follows:

```
$(function(){
    var span = $('.span-element');
});
```

Not only can we select elements based on some identifier we specify (that is, class or ID), but we can also select elements based on their tag name. If you wanted to select all the `li` elements within a page, you would use `$('li')`, illustrated as follows:

```
$(function(){
    var listelements = $('li');
    var i = 1;
    listelements.each(function(){
        console.log("Loop: " + i);
        i++;
    });
});
```

The preceding example uses the jQuery selector to select all the list elements within the page. To demonstrate that `listelements` now contains multiple elements, we loop through these and output some information to the console.



`.each()` is a jQuery function. Learn more about its uses in *Chapter 3, Loading and Manipulating Dynamic Content with AJAX and JSON*.

The console output for the preceding example is as follows:

```
Loop: 1
Loop: 2
Loop: 3
```



You can access the JavaScript console in various ways depending on your choice of browser:

- ▶ **Chrome:** *Ctrl + Shift + J* (**Mac:** *command + option + J*)
- ▶ **Internet Explorer:** *F12*
- ▶ **Firefox:** *Ctrl + Shift + K*

There's more...

It is also possible to select elements based on other properties such as their `rel` or `disabled` attributes.

The following code shows us how we can select an anchor element that has a `rel` attribute of `nofollow`:

```
$(function(){
    var nofollow = $('a[rel="nofollow"]');
});
```

See also

- *Finding and selecting sibling elements*

Finding and selecting sibling elements

You may not always know the specific element that you need to select. You may only know its parent, and therefore, you will need to search through the elements within the parent in order to find the specific element that you are looking for. This recipe will show you how to find elements through their parents in various ways.

Getting ready

Open your text editor or IDE with the latest version of jQuery, ready to be included into the HTML page that you will create as part of this recipe.

How to do it...

To learn the various ways in which jQuery can help you to search for DOM elements based on a parent element, perform each of the following steps:

1. Create a web page with the following HTML and JavaScript code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Finding and selecting sibling elements</title>
    <script src="jquery.min.js"></script>
    <script>
        $(function(){
            var element1 = $('#content .top .top-left'); //Select the
            top left division element
```

```
        var element2 = $('parent').find('a'); //Select the
anchor element
        var element3 = $('parent').find('grandchild');
        //Select the grandchild element
    });
</script>
</head>
<body>
<div class="division-container">Some text <span>within</span> a
div <span>which</span> has a many <span>span</span> elements.</
div>
<div id="content">
    <div class="top">
        <div class="top-left">Left</div>
        <div class="top-right">Right</div>
    </div>
</div>
<ul class="info-list">
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
</ul>
<ul class="second-info-list">
    <li>Second List Item 1</li>
    <li>Second List Item 2</li>
    <li>Second List Item 3</li>
</ul>
<div class="parent">
    <div class="child">
        <div class="grandchild">
            <a href="#">A Link</a>
        </div>
    </div>
</div>
</body>
</html>
```

2. This code uses multiple class names in the same way as you would use them with CSS to select child elements from HTML. Alternatively, you can use jQuery's `find()` function on a parent element to search within.

How it works...

The simplest way to select a child element based on its parent is by using the same selectors as you would in CSS (that is, `.classname .anotherclass`). Having said this, you do not always know the exact location of the sibling element you are looking for. If this is the case, we can use the useful jQuery's `find()` function. jQuery's `find()` function will search within the specified parent element for the sibling element that you are looking for.

Based on the HTML within the *How to do it...* section, the following JavaScript illustrates how you can access a child element directly in the same manner as you would in CSS:

```
$(function(){
    var element1 = $('#content .top .top-left');
});
```

This would make the DOM element available within the `content` variable. More on what this means is covered later in the chapter.

To find a child element without knowing its exact location, we can use the following JavaScript to locate the anchor within the `<div class="grandchild">` element:

```
$(function(){
    var element2 = $('.parent').find('a');
});
```

Note that you only need to specify the parent selector and the element you are looking for. The `find()` method simply traverses the DOM based on the specified parent element until it either finds the element you are looking for or runs out of elements to check against. You can use ID and class names within the `find()` method as well as HTML notations.

There's more...

You can also use CSS3 selectors such as `:first-child` and `:last-child` within `$()` to help you select the required DOM element.

See also

- *Selecting elements*

Creating DOM elements

To create rich and interactive user interfaces, we need to be able to dynamically add DOM elements to a web page. Elements may need to be added to a web page based on user interaction or another event such as page load.

Getting ready

For this recipe, you are going to need another blank HTML file. Create a new HTML file named `recipe-3.html` within the same directory as the one used for the previous recipe's files.

How to do it...

Learn how to create DOM elements with jQuery by performing the following steps:

1. Add the following HTML code to your `recipe-3.html` file in order to create a basic HTML page with an unordered list and include the jQuery library:

```
<!DOCTYPE html>
<html>
<head>
  <title>Creating DOM elements</title>
  <script src="jquery.min.js"></script>
  <script></script>
</head>
<body>
<div id="container">
  <ul id="myList">
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
  </ul>
</div>
</body>
</html>
```

2. Add the following JavaScript within the script tags in the head of the HTML document. The following JavaScript code will add two buttons to the DOM after the `#myList` element utilizes jQuery's `after()` and `insertAfter()` functions:

```
$(function() {
  $('#myList').after("<button>Button 1</button>");
  $('<button>Button 2</button>').insertAfter("#myList");
});
```

How it works...

To dynamically add DOM elements to any part of the document, we can use the `append()`, `addAfter()`, `after()`, `addBefore()`, and `before()` functions of jQuery. The functions `after()` and `insertAfter()` essentially perform the same action; the difference lies in the order in which the expressions are specified. This is the same for `insertBefore()` and `before()`.

Based on the HTML file in the *How to do it...* section, the following JavaScript will add two button elements after the unordered list element:

```
$(function() {
    $('#myList').after("<button>Button 1</button>");
    $('<button>Button 2</button>').insertAfter("#myList");
});
```

Once the preceding JavaScript has been executed, the HTML rendered in the browser should be modified as follows:

```
<!DOCTYPE html>
<html>
<head>
    <title> Creating DOM elements</title>
</head>
<body>
<div id="container">
    <ul id="myList">
        <li>List Item 1</li>
        <li>List Item 2</li>
        <li>List Item 3</li>
    </ul>
    <button>Button 2</button>
    <button>Button 1</button>
</div>
</body>
</html>
```

Note that even though the second button was added last, it is first in the HTML. This is because we have specified that the button should be inserted after the unordered list element. Both `.before()` and `.insertBefore()` jQuery methods work exactly in the same way, except that the button elements would be above the unordered list element.

A common requirement of dynamic web pages and web applications is to be able to add new items to a list. This is best achieved using the `.append()` function:

```
$(function() {  
    $('#myList').append("<li>List Item 4</li>");  
});
```

This JavaScript will add the new list item with the text `List Item 4` to the bottom of the `#myList` unordered list element. Alternatively, the `prepend()` function could be used to insert the list item at the top of the list.

There's more...

jQuery provides developers with many ways to add, append, insert, and update elements into the DOM that could not be demonstrated within a single recipe. Ensure that you are aware of the alternatives by reading the jQuery documentation.

See also

- ▶ *Inserting content into an element*
- ▶ *Removing DOM elements*
- ▶ *Re-using DOM elements*

Inserting content into an element

Interactive and dynamic web applications and websites not only require the web developer to be able to create DOM elements but also require the developer to be able to add dynamic content. This is easily achievable with another set of jQuery functions.

Getting ready

Create a blank HTML document named `recipe-4.html`, and ensure that you have the latest version of jQuery available to be included within this HTML document.

How to do it...

Learn how to dynamically add content into the DOM by performing each of the following steps:

1. Add the following code to your newly created HTML document, which will create a simple HTML web page:

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

```

<head>
  <title>Insert content into an element</title>
  <script src="jquery.min.js"></script>
  <script>

    </script>
</head>
<body>
<div id="container">
  <p>Here is some current HTML content</p>
</div>
<textarea id="myTextarea"></textarea>
</body>
</html>

```

2. Insert the following JavaScript code within the script tags in the document head. This code will inject different HTML content and elements into the DOM at various points.

```

$(function() {
  //Remove the container elements current HTML
  $('#container').html("<p>I have replaced the all the HTML
within the #container element</p>");

  //Add some more HTML to the beginning of the container element
  $('#container').prepend("<p>Another paragraph that has been
prepended.</p>");

  //Add a button to the end of the container element after all
other HTML content
  $('#container').append("<button>A Button Appended</button>");

  //Add some text into the text area element
  $('#myTextarea').val("Added some text using .text()");
});

```

How it works...

The quickest way to add content to an element is to use the `html()` function. By providing this function with a string as an argument, it will replace the selected element's current DOM contents with the provided string. If no string is provided, this function will return the element's DOM contents formatted as an HTML string.

Besides replacing the content of an element, we can also use `append()` and `prepend()` to add additional content at the end and at the beginning of the current content, respectively. Additionally, we have other functions available such as `text()`, which will decode any HTML before it inserts the string within the element. The `text()` function is typically used for text areas for this reason.

Based on the HTML provided in the previous section, we can alter the content of the `#container` element using the jQuery functions previously discussed as follows:

```
$(function() {
  $('#container').html("<p>I have replaced the all the HTML within
    the #container element</p>");

  $('#container').prepend("<p>Another paragraph that has been
    prepended.</p>");

  $('#container').append("<button>A Button Appended</button>");

  $('#myTextarea').val("Added some text using .text()");
});
```

After each of these functions has been executed, the HTML file rendered by the browser will be transformed, which is illustrated as follows:

```
<!DOCTYPE html>
<html>
<head>
  <title>Insert content into an element</title>
</head>
<body>
<div id="container">
  <p>Another paragraph that has been prepended.</p><p>I have
    replaced the all the HTML within the #container element</p>
  <button>A Button Appended</button>
</div>
<textarea id="myTextarea">Added some text using .text()</textarea>
</body>
</html>
```

See also

- *Creating DOM elements*

Modifying the DOM element properties

We can use jQuery to dynamically modify element properties such as class, style, and disabled, which means that it is possible to visually alter and change the function of a range of HTML elements.

Getting ready

Once again, this recipe requires an additional blank HTML document. Create a file named `recipe-5.html`, and have it open and ready for editing.

How to do it...

Learn how to alter the properties of the DOM element by performing each of the following steps:

1. Add the following HTML code to your blank `recipe-5.html` file in order to create a basic HTML page with two types of inputs:

```
<!DOCTYPE html>
<html>
<head>
  <title>Modifying DOM element attributes and properties</title>
  <script src="jquery.min.js"></script>
  <script>
    </script>
</head>
<body>
  <input type="checkbox" />
  <input type="text" />
</body>
</html>
```

2. Within the preceding HTML code, add the following JavaScript code inside the script tags to disable the input, modify its value, and check the checkbox:

```
$(function(){
  //Set the checkbox to be checked
  $('input[type="checkbox"]').prop('checked', true);
  //Disable any text inputs
  $('input[type="text"]').prop('disabled', true);
  //Change the value of any text inputs
  $('input[type="text"]').val("This is a new Value!");
});
```

How it works...

jQuery provides us with a `prop()` function that will either retrieve the specified property if no value is specified, or if a value is provided, it will alter the specified property on the selected element. This can be used to change property values such as `checked` on a checkbox or the `disabled` property on a text input. We could use the `prop()` function to alter the value of a text input; however, it is preferable to use the `val()` function that is available specifically for this task.

Typically, this would be done based on a user-triggered event, but to illustrate this as simply as possible, the following JavaScript does so on page load:

```
$(function() {  
    $('input[type="checkbox"]').prop('checked', true);  
});
```

This JavaScript will check each input within the page that is of the type `checkbox`. Similarly, we can alter the disabled state of a text input with only a few modifications:

```
$(function() {  
    $('input[type="text"]').prop('disabled', true);  
});
```

We can also use the `val()` function to add some text to each of these text inputs using the following JavaScript:

```
$(function() {  
    $('input[type="text"]').val("This is a new Value!");  
});
```

Often, you can chain functions with jQuery. You can achieve the previous two actions by using both the functions inline (that is, `$('input[type="text"]').prop('disabled', true).val("This is a new Value!");`), and they will be executed in turn.

See also

- ▶ *Enabling and disabling buttons by changing their properties*
- ▶ *Adding and removing CSS classes to dynamically change their style*

Adding and removing CSS classes to dynamically change their style

jQuery comes bundled with class manipulation functions in order to allow developers to easily alter the style of any HTML element.

Getting ready

For element style changes to be of any use, we first need to declare some styles within an HTML document. The following HTML code has a range of styles and elements that we can work with to illustrate this functionality of jQuery:

```
<!DOCTYPE html>
<html>
<head>
  <title>Add and remove CSS classes to dynamically change their
    style</title>
  <script src="jquery.min.js"></script>
  <script></script>
  <style type="text/css">
    .green {
      background-color: #008000;
      color: #FFFFFF;
    }
    .red {
      background-color: #FF0000;
      color: #FFFFFF;
    }
    .yellow {
      background-color: #FFFF00;
      color: #000000;
    }
  </style>
</head>
<body>
  <p id="sometext">
    Here is some text that can have different styles applied to
    it dynamically</p>
  <button id="green-btn">Green</button>
  <button id="red-btn">Red</button>
  <button id="yellow-btn">Yellow</button>
</body>
</html>
```

Within this HTML code, we have three buttons with their own unique IDs. We also have a paragraph with an ID. There are three CSS classes defined: `green`, `red`, and `yellow`. With jQuery, we can listen for the click of either of these buttons and then dynamically apply one of these classes to the paragraph element.

If you save this HTML file and open it within a browser, you should have the following web page:

Here is some text that can have different styles applied to it dynamically

Green

Red

Yellow

How to do it...

1. Add the following JavaScript code within the script tags in the HTML page you have just created:

```
$(function(){  
    //Listen for a click event on the green button  
    $('#green-btn').click(function(){  
        //When the green button has been clicked  
        //Remove all classes current on the #sometext paragraph  
        $('#sometext').removeClass();  
        //Add the .green class to the #sometext paragraph  
        $('#sometext').addClass('green');  
    });  
    //Listen for a click on the red button  
    $('#red-btn').click(function(){  
        //When the red button has been clicked  
        //Remove all classes from the #sometext paragraph  
        $('#sometext').removeClass();  
        //Add the .red class to the #sometext paragraph  
        $('#sometext').addClass('red');  
    });  
    //Listen for a click on the yellow button  
    $('#yellow-btn').click(function(){  
        //When the yellow button has been clicked  
        //Remove all classes from the #sometext paragraph  
        $('#sometext').removeClass();  
        //Add the .yellow class to the #sometext paragraph  
        $('#sometext').addClass('yellow');  
    });  
});
```

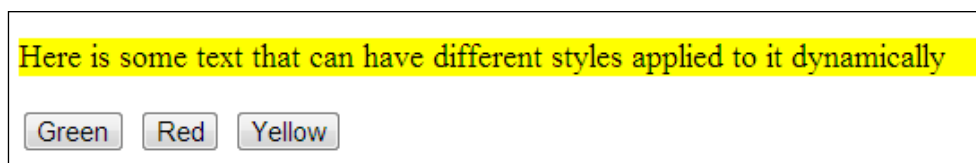
2. Opening the HTML document in your browser will now allow you to change the #sometext paragraph style by selecting either of the three available buttons.

How it works...

jQuery allows us to attach a click event handler to any element by using the `click()` function. We can then execute a set of code of our choice by passing a function as an argument to the `click()` method. To add a class to an element, we can use the `addClass()` function and provide the class name as a string argument. This function will add the specified class name to the selected element.

jQuery also provides us with a `removeClass()` function. This allows us to either remove a specific class from an element by providing `removeClass()` with a string, or when a string is not provided, it will remove all the classes from the selected element. We will need to use this in order to prevent multiple classes being added to the paragraph element when either of the buttons has been clicked more than once.

The following screenshot illustrates this web page after the **Yellow** button has been clicked:



See also

- ▶ *Modifying the DOM element properties*
- ▶ *Enabling and disabling buttons by changing their properties*

Enabling and disabling buttons by changing their properties

The ability to dynamically enable and disable buttons is particularly useful for situations such as saving data to a web server. In order to prevent a user from making multiple save requests while the request is being made and the client is waiting for a response, you can dynamically disable the save button. Once the client has received a response from the web server, you can re-enable the save button.

This functionality can also be very effective in simple situations, such as enabling the search button when the user has inputted a search term. This makes it clear to the user that they cannot search unless a term has been entered.

Getting ready

Create a blank HTML document named `recipe-7.html`, and have it open and ready for editing.

How to do it...

1. The following HTML code creates a web page with a search input and a search button, which is disabled by default. Add the following code to `recipe-7.html`:

```
<!DOCTYPE html>
<html>
<head>
  <title>Enable and disable buttons by changing their
    properties </title>
  <script src="jquery.min.js"></script>
  <script>

    </script>
</head>
<body>
  <input type="text" id="search-input" />
  <button id="search-btn" disabled>Search</button>
</body>
</html>
```

2. Saving and opening this HTML in a browser should provide you with a very simple web page having a single input and a disabled button as illustrated in the following screenshot:



3. Add the following JavaScript within the script tags in the head section of the HTML document created previously:

```
$(function(){
  //Listen for a key up event on the search input
  $('#search-input').keyup(function(){
    //When a user presses and releases a key
    //Check to see if the length of the inputted
    //data is greater than 2
    if ($(this).val().length > 2) {
      //If the input length is greater than
```

```

        //two then we enable the search button
        $('#search-btn').prop("disabled", false);
    } else {
        //If the input length is equal to 2 or less we
        //disable the search button
        $('#search-btn').prop("disabled", true);
    }
});
});

```

4. Opening this page within a web browser will provide you with an input and a disabled search button until you enter some text into the search input. When text is entered into the search input and the length of the text is greater than two characters, the search button will become available.

How it works...

Our aim is to enable the search button once there has been some text inputted into the search input by the user. To do this, we need to attach a `.keyup()` event handler to the search input. This will allow us to execute some code while the user is inputting some text. By providing a function as an argument to the `keyup()` function, we can then check the inputted data. If the input data has a length of two or more characters (as a search less than three characters would be a little ambiguous), we can enable the search button.

Using the following JavaScript, we are able to listen for data input, check the input length, and depending on this, enable or disable the search button:

```

$(function() {
    $('#search-input').keyup(function() {
        if ($(this).val().length > 2) {
            $('#search-btn').prop("disabled", false);
        } else {
            $('#search-btn').prop("disabled", true);
        }
    });
});

```

First of all, we attach the `keyup` event to the search input using `$('#search-input').keyup()`, referencing its ID. Then, within the callback function, we are able to check the length of the currently inputted text using `$(this)`, which refers to the element to which we have attached the `keyup` event. The `val()` function then gets the inputted text, and we can use the `length` property to find its length. Using an `if/else` statement, we can decide if the search button needs to be enabled or disabled.

To enable or disable the search button, we use jQuery's `prop()` function and set the disabled property to either `true` or `false`.

See also

- ▶ *Modifying the DOM element properties*
- ▶ *Adding and removing CSS classes to dynamically change their style*

Updating an image within a page

jQuery allows the developer to dynamically change images on a web page. This recipe will show you how to do this and also show you how to use a timestamp in order to prevent the browser from using a cached image, which can often be a problem when swapping images dynamically in this way.

Getting ready

For this recipe, you are going to need four different images. Ensure that you have four small images named `black.png`, `red.png`, `blue.png`, and `green.png` available.

How to do it...

To understand how jQuery can be used to change an image, complete each of the following steps:

1. Create a file named `recipe-8.html` within an easily accessible directory, and add the following HTML code to this file:

```
<!DOCTYPE html>
<html>
<head>
  <title>Change an image source and tackle browser caching
    to ensure it is always updated</title>
  <script src="jquery.min.js"></script>
  <script>
    </script>
</head>
<body>
  
  <div>
    <button id="red-btn">Red</button>
    <button id="green-btn">Green</button>
    <button id="blue-btn">Blue</button>
  </div>
</body>
</html>
```

2. Within the directory where the `recipe-8.html` file is created, create another directory called `images` and within this, add four images given as follows:
 - ❑ `black.png`
 - ❑ `red.png`
 - ❑ `blue.png`
 - ❑ `green.png`
3. Add the following JavaScript within the `<script></script>` tags of `recipe-8.html`:

```
$(function(){  
    //Listen for a click on the red button  
    $('#red-btn').click(function(){  
        //When the red button has been clicked, change the source of  
        the #square image to be the red PNG  
        $('#square').prop("src", "images/red.png");  
    });  
    //Listen for a click on the green button  
    $('#green-btn').click(function(){  
        //When the green button has been clicked, change the source of  
        the #square image to be the green PNG  
        $('#square').prop("src", "images/green.png");  
    });  
    //Listen for a click on the blue button  
    $('#blue-btn').click(function(){  
        //When the blue button has been clicked, change the source of  
        the #square image to be the blue PNG  
        $('#square').prop("src", "images/blue.png");  
    });  
});
```
4. Opening this web page within a browser will allow you to change the source of the displayed image from the default `black.png` to another source depending on which button is clicked.

How it works...

To change the source of an image, we can use jQuery's `prop()` function and specify the new image name for the `src` property. To do this, when either of the buttons created using our HTML code are clicked, a click event handler is attached for each button using `.click()`, referencing the buttons' IDs, and then within the `click()` callback function, `.prop()` is executed with the appropriate image source specified, shown as follows:

```
$(function(){  
    $('#red-btn').click(function(){
```

```
    $('#square').prop("src", "images/red.png");
  });

$('#green-btn').click(function() {
    $('#square').prop("src", "images/green.png");
  });

$('#blue-btn').click(function() {
    $('#square').prop("src", "images/blue.png");
  });
});
```

There's more...

This recipe illustrates the way a jQuery developer can easily change an image's source using a very simple example. A more realistic situation where this implementation will be used is within a web application where an image can be uploaded, for example, when a user chooses their avatar.

Traditionally, a user will be presented with a preview of their current avatar and then be able to choose an image from their computer to upload. Using AJAX, the web page can send this new image to the server; the server can then process and save this image and respond to the client web page. The web page, using jQuery's `prop()` method, can then update the current preview with the newly uploaded image and create a seamless transition without the need for the page to be refreshed in order to display the new image.

A problem occurs when the server uses the same filename for the new image as the old one. This is often the case when a user can only have one avatar; for the sake of simplicity, the avatar image is then saved using the user's unique ID (for example, `123.png`).

When the server responds to the client with the new image filename, as the filename is the same, the browser will think that it is the same image. This may cause the browser to use the cached version of the avatar image, which will be the old image. To prevent this from happening, we can prepend a timestamp onto the image's filename. This will make the browser treat the image as new and force it to load the new image. We can modify the previous JavaScript to achieve the following:

```
$(function() {
    $('#red-btn').click(function() {
        $('#square').prop("src", "images/red.png?t=" + new Date().
        getTime());
    });

    $('#green-btn').click(function() {
        $('#square').prop("src", "images/green.png?t=" + new Date().
        getTime());
    });
});
```

```

});

$('#blue-btn').click(function() {
    $('#square').prop("src", "images/blue.png?t=" + new Date().
getTime());
});
});

```

Using JavaScript's new `Date()` method, we create a new date that will be equal to the current date and time equal to the current time in milliseconds. We then use `.getTime()` to return a timestamp in milliseconds. When the source is updated, it will look as follows:

```

```

This code will force the browser to reload the image using the newly specified source, provided the user does not update their image within the same millisecond (practically impossible).

Populating list elements

List elements are commonly used around the Web; they can be used to display search results, menus, and navigational items to name a few. Thanks to CSS, they no longer need to be boring, and it is possible to style list elements to make your data beautiful.

With jQuery, it is possible to populate a list element dynamically. This can be done directly from a JavaScript array via an AJAX response, with data from a web server or some other source.

Getting ready

Create a blank HTML document named `recipe-9.html`, and ensure that it is saved to a location where the latest version of jQuery can be included.

How to do it...

Learn how to dynamically populate a list with jQuery by performing each of the following recipes:


1. In order to demonstrate how you can use jQuery to populate a list element, we will create a JavaScript array of objects. Add the following HTML and JavaScript code to `recipe-9.html`, which you have just created:

```

<!DOCTYPE html>
<html>
<head>
    <title>Populating list elements</title>
    <script src="jquery.min.js"></script>

```

```
<script type="text/javascript">
  var names = [
    {
      id: 1,
      firstname: 'Leon',
      lastname: 'Revill'
    },
    {
      id: 2,
      firstname: 'Allyce',
      lastname: 'Wolverson'
    },
    {
      id: 3,
      firstname: 'Harry',
      lastname: 'Round'
    },
    {
      id: 4,
      firstname: 'Chris',
      lastname: 'Wilshaw'
    }
  ];
  $(function(){
  });
</script>
</head>
<body>
  <ul id="namelist"></ul>
</body>
</html>
```

 At the top of our JavaScript code, we have created an array of objects which includes a set of names. We are going to use this array to populate the list element #namelist within the HTML code.

2. Add the following JavaScript within `$(function() {});`, just under the JavaScript array. This JavaScript will use the objects within the JavaScript array we created in the *Getting ready* section to populate the list element on our page.

```
$.each(names, function(index, obj){
  $('#namelist').append("<li>#" + obj.id + " " +
    obj.firstname + " " + obj.lastname + "</li>");
});
```

How it works...

We use jQuery's `$.each()` function to loop through each of the JavaScript objects within the `names` array. Then, for each of these objects, we can create a `` element and insert the values of the `id`, `firstname`, and `lastname` variables. Finally, we can use the jQuery `append()` function to append the list element to the end of the unordered list.

Within the `$.each()` function, the first parameter is the array we wish to iterate through and the second parameter is the function we wish to execute for each of the objects within the `names` array. The specified function also has two arguments: `index` and `obj`. The `index` argument will contain the current array index of the JavaScript object, and the `obj` variable will contain the actual JavaScript object. Both these variables are available within the specified callback function.

We are then able to reference `obj.propertyName` (replace `propertyName` with a property of the object) in order to access specific parts of the object we wish to use. By doing this, we construct a string and pass it to the `append()` function, which then appends it to the specified `#nameslist` unordered list.

Open the HTML page within the browser, and you should see the list populated with the names from the JavaScript array, illustrated as follows:

- #1 Leon Revall
- #2 Allyce Wolverson
- #3 Harry Round
- #4 Chris Wilshaw

See also

- ▶ *Creating DOM elements*
- ▶ *Re-using DOM elements*

Understanding pagination

Pagination is the act of collating large amounts of data and presenting it to the user in small, easy-to-read sections or pages.

With a combination of jQuery, JavaScript functions, and event handlers, we are able to easily collate and present data to the user in pages.

Getting ready

To create a paginated set of data, we first need some data to paginate and then a location to place the paginated data. Use the following code to create an HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <title>Chapter 1 :: DOM Manipulation</title>
  <script src="jquery.min.js"></script>
  <script>
    var animals = [
      {
        id: 1,
        name: 'Dog',
        type: 'Mammal'
      },
      {
        id: 2,
        name: 'Cat',
        type: 'Mammal'
      },
      {
        id: 3,
        name: 'Goat',
        type: 'Mammal'
      },
      {
        id: 4,
        name: 'Lizard',
        type: 'Reptile'
      },
      {
        id: 5,
        name: 'Frog',
        type: 'Amphibian'
      },
      {
        id: 6,
        name: 'Spider',
        type: 'Arachnid'
      },
      {
        id: 7,
```